Simplificació de models de malles complexes.

Treball de final de grau

Autor: Martí Sala Morral

Director: Antonio Chica Calaf

Grau en Enginyeria Informàtica
Especialitat de computació

Facultat d'Informàtica de Barcelona (FIB)
Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

30 de gener
Curs 2018-19 Q1

Resum (Anglès)

The mesh simplification problem is a problem from the computer graphics field. There's been a lot of work on this problem and there are so much algorithms that allow us to generate good simplifications for a lot of mesh models.

In this project we'll focus our work on searching an algorithm capable of generate good simplifications for models with complex topology and/or geometry because the current algorithms have huge trouble dealing with them.

In order to do so, we'll work with surface reconstruction and simplification algorithms and we'll create a pipeline between them. Our goal is to create a wrapping surface of the original model good enough to simplificate it instead of the model. So the simplification of the surface will be the simplification of the model.

Finally, in order to achieve more visual similarity between the original mesh and the simplified one, we'll do some postwork in order to transfer properties of the original mesh to the simplified one such as normals or textures.

Resum (Català)

El problema de la simplificació de models de malles és un problema del camp de gràfics per computador. És un problema tractat àmpliament i existeixen algorismes que ens permeten generar simplificacions correctes per a molts models de malles.

En aquest projecte en centrem en buscar un algorisme que sigui capaç de produir bones simplificacions per a models de malles triangulars que tinguin geometries i/o topologies complexes, ja que són els models que causen més problemes als algorismes actuals.

Per a fer-ho treballarem amb algorismes de reconstrucció de superfícies i algorismes de simplificació de malles creant un *pipeline* entre aquests. El nostre objectiu és aconseguir una superfície envolupant del model original que sigui prou fidel a aquest per a que després la poguem simplificar i utilitzar com a model simplificat de l'original.

Finalment, afegim un postprocessat per a transferir propietats de la malla original, com les normals o textures, a la simplificada aconseguint així més similitud visual entre elles.

Resum (Castellà)

El problema de la simplificación de modelos de mallas és un problema del campo de gráficos por computador. És un problema ampliamente tratado y existen algoritmos que nos permiten generar simplificaciones correctas para muchos modelos de mallas.

En este proyecto nos centramos en buscar un algoritmo que sea capaz de producir buenas simplificaciones para modelos de mallas triangulares que tengan geometrías y/o topologías complejas, ya que son los modelos que causan más problemas a los algoritmos actuales.

Para hacerlo trabajamos con algoritmos de reconstrucción de superfícies y algoritmos de simplificación de mallas creando un *pipeline* entre estos. Nuestro objetivo es conseguir una superfície envolvente del modelo original que sea suficientemente fiel al mismo para después poderla simplificar y utilizarla como modelo simplificada del original.

Finalmente, añadimos un postprocesado para transferir propiedades de la malla original, como las normales o texturas, a la simplificada consiguiendo así más similitud visual entre ellas.

Índex

1. Introducció

Una gran quantitat de les aplicacions gràfiques que utilitzem actualment representen objectes en tres dimensions. Molt sovint es vol que aquests objectes siguin tant detallats o fidels als models reals com sigui possible.

Però per a poder representar models més detallats o fidels es requereix més capacitat de càlcul, afectant així al rendiment de les aplicacions en qüestió. Aquest problema ha estat atacat des de fa temps a través de les tècniques que coneixem com a tècniques de simplificació de malles. Sense entrar en detall, aquestes tècniques prenen com entrada un model detallat i en retornen un de més simple.

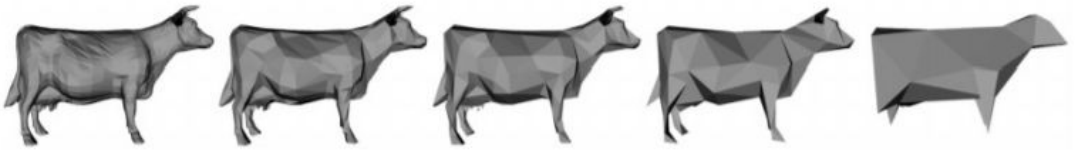


Figura 1.1. Model original seguit d'una seqüència de models cada cop més simples [2].

Aquests models simplificats s'utilitzen per a representar els models més detallats quan les circumstàncies ho requereixen o ho permeten. Per exemple, els elements més allunyats de l'escena no requereixen tant de detall com els elements més propers o en primer pla. Així les aplicacions aconsegueixen estalviar cost computacional per a representar els models de l'escena.

Figura 1.2. Captura del videojoc battlefield 1.

A la figura anterior podem observar com els objectes més propers a l'escena i en primer pla són més detallats, mentre que els fons de l'escena i els objectes més allunyats cada cop tenen menys detall. Les aplicacions interactives com els videojocs són un bon exemple ja que utilitzen tot tipus de tècniques per a estalviar recursos a l'hora de pintar l'escena.

No obstant, aquestes tècniques erren amb certs tipus de models. Per exemple, els que tenen topologies o formes complexes, obtenint resultats clarament inacceptables com a simplificacions dels models originals.

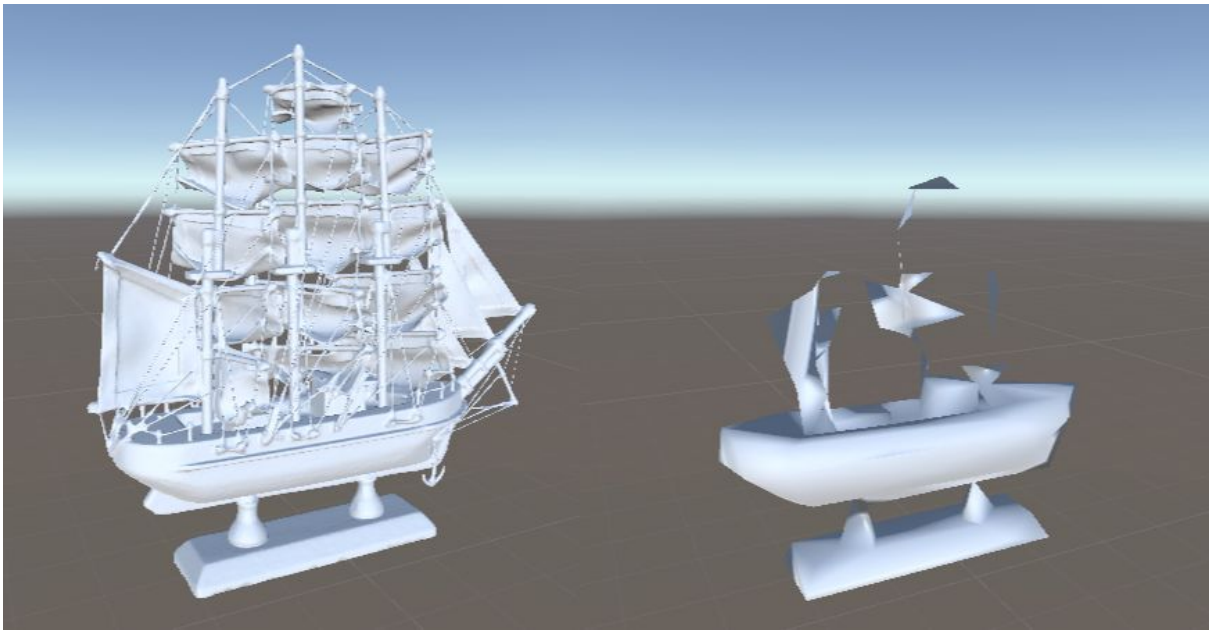


Figura 1.3. A l'esquerra: model original. A la dreta: simplificació del model original.

No entrarem en els detalls aquí però, per a contextualitzar, la simplificació de la figura anterior està feta amb el mètode *quadric edge collapse decimation*, un dels mètodes més emprats actualment. Podem observar com a la simplificació ha "desaparegut" bona part de la geometria, perdent així bona part de la similitud amb el model original.

Així doncs el problema en el que ens centrarem consisteix en cercar una tècnica que ens permeti obtenir models simplificats d'aquests models i que les tècniques actuals no poden simplificar amb resultats acceptables.

2. Conceptes

Per a poder seguir aquesta memòria es requereixen certs coneixements sobre representació d'objectes a través de models, com treballar amb aquestes representacions i reconstrucció de superfícies. En aquest apartat explicarem tots aquests conceptes que a la resta de la memòria és donaran per suposats.

2.1 Com representar objectes

Per a poder representar els objectes en aplicacions de tres dimensions s'utilitzen models. Aquests models els podem dividir en dues categories, els models volumètrics o sòlids o els models de superfície. En aquest projecte ens centrarem en els models de superfície.

Els models de superfície, o poligonals, són aquells models que estan formats per un conjunt de punts a l'espai tridimensional. Aquest conjunt de punts s'encarrega de definir la geometria del model, és a dir la seva forma a l'espai. A causa de la seva estructura els models de superfície també s'anomenen models de malles.

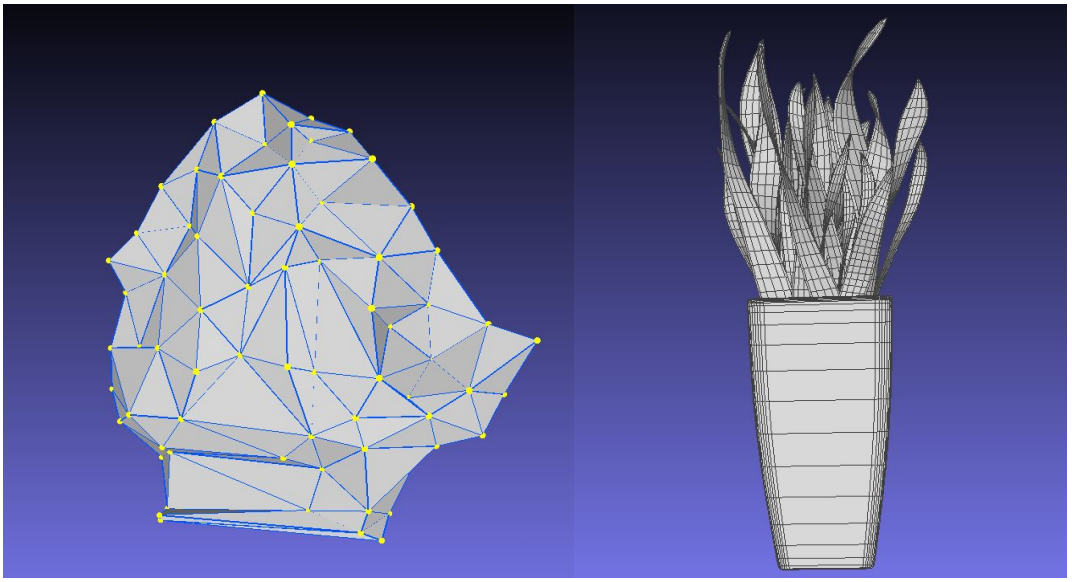


Figura 2.1. A l'esquerra: Model de malla triangular. A la dreta: Model de malla amb quads.

Apart d'aquests punts, els models de superfície estan formats per un conjunt d'arestes que relacionen els punts entre ells. Les arestes defineixen les topologia del model, indicant quines parts del model són forats buits o formen part de la superfície del model.

Cada part de superfície delimitada per un conjunt d'arestes és una cara del model. Cada cara del model té la seva part interior a la superfície i la seva part exterior. Tot i que actualment una gran majoria de les malles es representen amb cares triangulars. També existeixen malles formades per altres polígons com per exemple *quads* o quadrats.

A la part esquerra de la figura 2.1 podem veure els components esmentats anteriorment. Els punts grocs representen els vèrtex, les línies blaves les arestes i les parts en diferents tons de gris les cares del model.

Per conveni, totes les cares estan orientades de forma coherent, de manera que la part exterior d'una cara coincideix amb la resta de cares adjacents, i les seves parts interiors no siguin visibles a l'escena. Les cares interiors les podem identificar fàcilment ja que es representen amb cares fosques que mai reven il·luminació.

També existeixen models on no és possible d'aconseguir orientar totes les cares de forma coherent, a la següent figura en podem veure exemples.

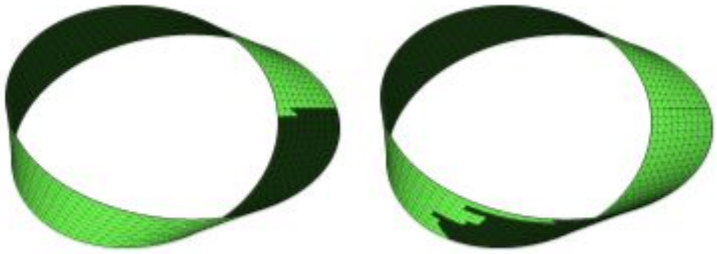


Figura 2.2. Models Mönibus, exemples de models on les cares no es poden orientar de forma coherent [11].

2.2 Treballar amb malles

En aquest projecte només treballarem amb malles formades per triangles. Tot i així les malles formades per *quads* o altre polígons es poden convertir fàcilment a malles formades per triangles a través d'una triangulació del polígon en qüestió.

2.2.1. Models *manifold* i *non-manifold*

els models de malles es poden dividir en dues categories , *manifold* i *non-manifold*, segons si compleixen un seguit de propietats. Els models manifold són aquells models que compleixen les següents condicions:

- Les cares només intersecten en vèrtex o arestes de la malla.
- El model no té elements inconnexos o conectats només per un vèrtex o una aresta que uneix un vèrtex de cada component.
- A cada aresta sempre hi intersecten exactament dues cares.

En canvi, els models *non-manifold* són tots aquells models que no compleixen les condicions anteriors. Això els permet tenir una topologia molt més flexible però fa que sigui més complicat treballar-hi.

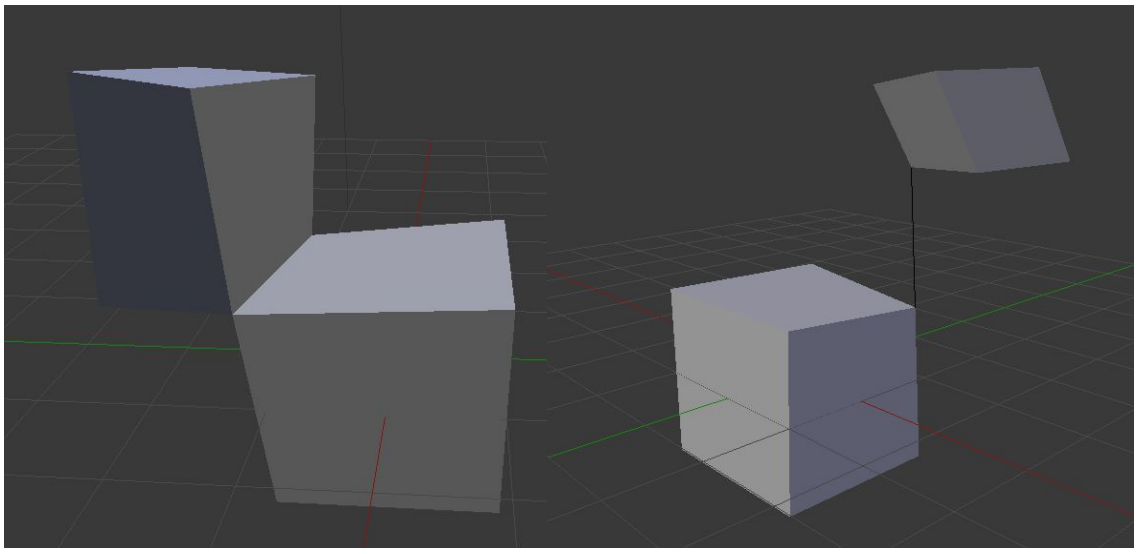


Figura 2.3. Exemples de models non-manifold. A l'esquerra hi ha una aresta on intersecten més de dues cares. A la dreta components de la malla connectats només per una aresta.

Molts dels algorismes existents estan pensats per a models *manifold* a causa de la dificultat que causa tractar amb models non-manifold. Per exemple, en el cas de la simplificació de malles, els algorismes que volen preservar la topologia tenen més problemes per a tractar models *non-manifold*.

2.2.3. Desplegat d'un model

El concepte de desplegar un model, o *unwrap* en anglès, consisteix en distribuir tota la superfície d'un model tridimensional en un pla. Això es fa per a poder fer una mapeix d'una textura o normal map, amb coordenades bidimensionals, sobre la superfície del model.

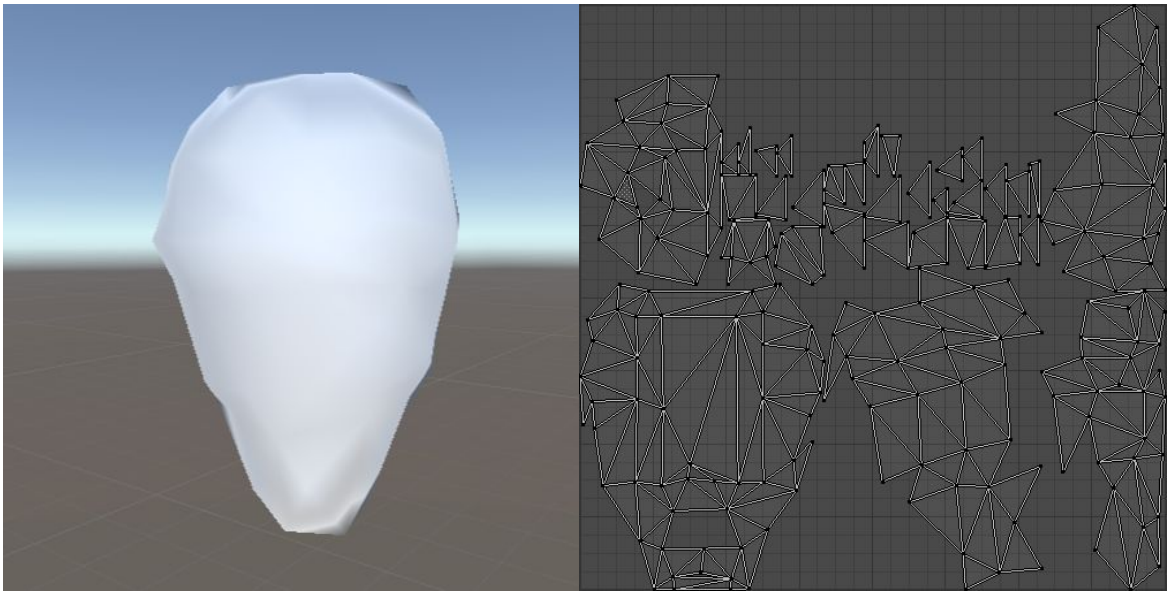


Figura 2.4. A l'esquerra: model de malla. A la dreta: desplegat del model.

Tot i que podem fer desplegats de models a mà, existeixen tècniques que automatitzen aquest procediment.

Per aconseguir-ho es realitza una projecció de la superfície sobre un pla. Existeixen tot tipus de projeccions, cada una addient per a tipus de models diferents. Així per exemple la projecció esfèrica aconsegueix millors resultats en models que tenen una forma semblant a una esfera. En canvi, la projecció cúbica obté bons resultats en models amb formes similars al cub.

2.2.4. *Baking*

El *baking* és una tècnica que consisteix en fer una imatge bidimensional de certes propietats de la superfície d'una malla per a poder estalviar temps en el pas de renderitzat, tals com la il·luminació o *normal maps*. Apart, per a poder aplicar la imatge a la superfície de la malla, aquesta a d'haver-se desplegat prèviament i tenir coordenades de textura.

Un cop s'han creat aquestes imatges, és fàcil transferir-les a altres models passant de les coordenades d'un model a l'altre. Així, per exemple, un model de baixa qualitat amb menys polígons pot mapejar a la seva superfície les normals del mateix model d'alta definició a través del *normal map* calculat a través de *baking.*

2.2.4. *Convex hull* d'un model

La *convex hull* d'un model 3D es defineix com la superfície envolupant convexa d'aquest model. És a dir, la superfície que conté tots els punts del model original i que no conté regions còncaves.

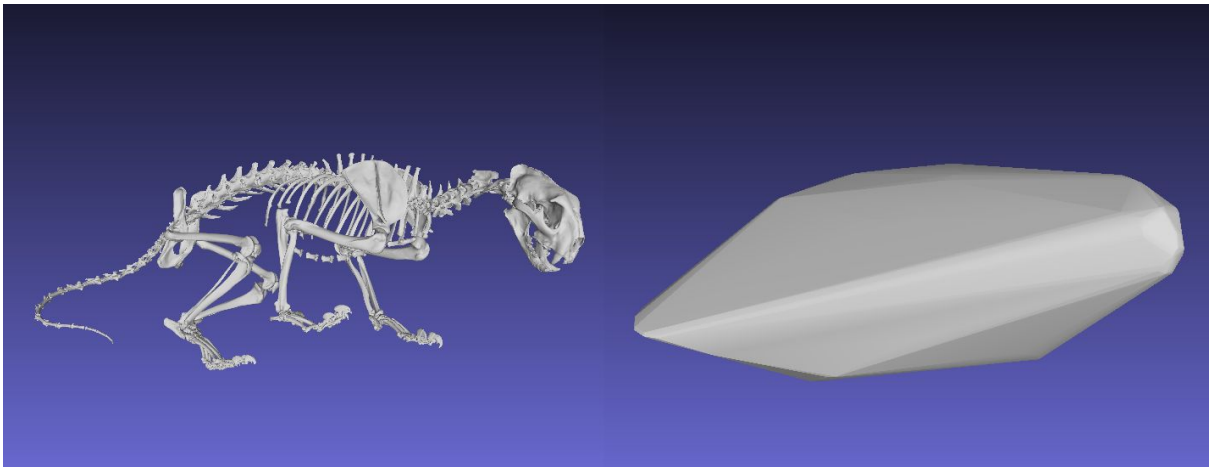


Figura 2.5. A l'esquerra: model original. A la dreta: convex hull del model.

Així doncs, la *convex hull* d'un model de malles és una superfície molt simple però perd tots els detalls que el model original tingués, així com tota la seva topologia. Tal com podem veure a la figura 2.5.

2.2.5. Nivell de detall

Les tècniques de nivell de detall o LOD, de les seves sigles en anglès *Level of Detail*, són tècniques per a millorar el rendiment de les aplicacions gràfiques. Per a fer-ho, utilitzen diversos models de malles que representen un mateix objecte amb diferents nivells de detall, d'aquí el nom de la tècnica.

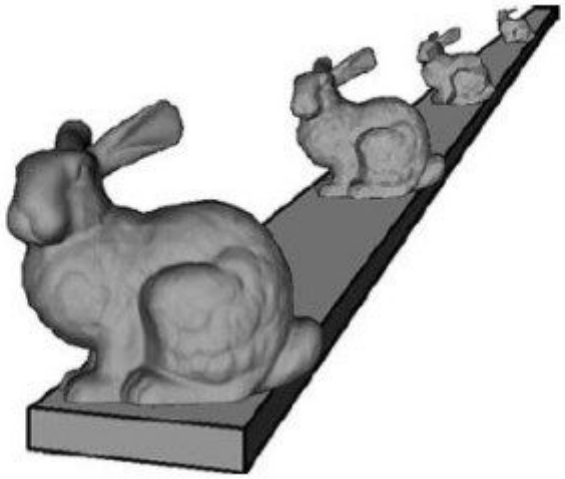


Figura 2.6. Nivells de detall diferents segons la distància al model.

Un cop es tenen definits els diferents nivells de detall d'un objecte a través d'aquests models de malles, es defineix a partir de quina distància s'utilitza cada model. Per exemple, el model més detallat s'utilitzarà quan aquest estigui més a prop de l'escena o en primer pla i el model amb menys detall s'utilitzarà quan l'objecte estigui al fons de l'escena.

Així, quan un objecte és poc important per a l'escena no haurem de processar tota la seva geometria per a representar-lo, sinó una versió reduïda. Reduint així el cost computacional de pintar l'objecte a l'escena.

A l'hora de definir els nivells de detall ens trobem amb dues vessants d'aquesta tècnica, la discreta i la contínua.

La vessant discreta parteix d'un conjunt de models ja creat prèviament que defineixen els nivells de detall. Llavors només resta decidir en quin moment és adient aplicar cada nivell de detall. Aquesta tècnica és més fàcil d'implementar i s'ajusta millor al *hardware* gràfic actual [18].

En canvi, la vessant contínua es basa en construir una estructura de dades que conté una seqüència contínua de nivells de detall. Un exemple és el *Progressive Meshes*, on l'estructura de dades que s'utilitza és una cadena d'operacions de simplificació *edge collapse* sobre la malla base [19].

2.2.6. Tipus de fitxers

Per a poder treballar amb els models de malles requerim fitxers que ens permetin emmagatzemar la informació d'un model per a poder ser llegida i processada en altres moments. Existeixen diversos tipus de fitxers per a representar models de malles àmpliament adoptats i utilitzats. Per exemple els tipus *wavefront* (.obj) o Stl (.stl). Durant el projecte hem treballat amb aquests dos tipus de fitxers però no és difícil convertir un model d'un tipus a un altre a través de programari tal com *Blender* o *MeshLab*.

Tots els tipus de fitxer emmagatzemen la informació sobre la topologia i geometria del model. Guardant les cares a partir dels vèrtex que les formen. Apart, es poden guardar altres dades rellevants, com les coordenades de textura o normals dels vèrtex i cares.

El tipus de fitxer *wavefront* és el sistema més utilitzat actualment. Permet emmagatzemar els vèrtex que formen la geometria, les coordenades de textura i normals del vèrtex, així com les cares que formen el model.

Els fitxers de tipus Stl tenen una estructura interna senzilla i en format ASCII permet treballar molt fàcilment amb aquest tipus de fitxer.

Un fitxer .stl en format ASCII sempre comença amb la línia *solid* i acaba amb la línia *endsolid*. Entremig es defineixen el conjunt de cares que conformen el model. Per cada cara, s'especifiquen els tres vèrtex que la defineixen i addicionalment es pot especificar el vector normal de la cara o simplement deixar-lo a zero ja que la majoria de software és capaç de calcular la direcció del vector normal d'un cara automàticament. Per a fer-ho s'escriuen les següent línies per a cada cara a representar:

facet normal n_i n_j n_k
  outer loop
    vertex $v1_x$ $v1_y$ $v1_z$
    vertex $v2_x$ $v2_y$ $v2_z$
    vertex $v3_x$ $v3_y$ $v3_z$
  endloop
endfacet

Aquest tipus de fitxer no permet representar cap altre propietat dels vèrtex tals com les coordenades de textura, cosa que el tipus *wavefront* si que permet.

2.2.6. Llibreries

Per a treballar de forma eficient i robusta amb models de malles existeixen diverses llibreries de codi obert que inclouen tipus de dades per a tractar els models de malles així com tot de mètodes i algorismes que existeixen actualment.

Nosaltres hem utilitzat la llibreria *alphashape3D* de R que conté tot el necessari per aplicar l'algorisme d'*alpha shape* a un conjunt de punts i la llibreria "rgl: 3D visualization using OpenGL"[14], que conté tot de mètodes per treballar amb fitxers de models de malles i models de malles.

Les llibreries més populars són, però, *CGAL (Computation Geometric Algorithms Library)* i *OpenMesh*. Es tracta d'un parell de llibreries en C++ que inclouen les estructures de dades necessàries per a treballar amb models de malles, així com implementacions dels algorismes que hem emprat en aquest projecte, entre d'altres tècniques i algorismes[6][7].

2.2.7. Programari de processat de malles

Actualment existeixen molts programes per a treballar amb models de malles. Entre els més populars trobem el programa *Blender*. Apart, nosaltres usarem altres programes no tan estesos com el *MeshLab* i *Graphite*.

Blender va ser publicat el 1995 com un programa per a poder tractar els primers fitxers de models de malles. El 2002 la companyia distribuïdora va fer fallida i el producte es va discontinuar.

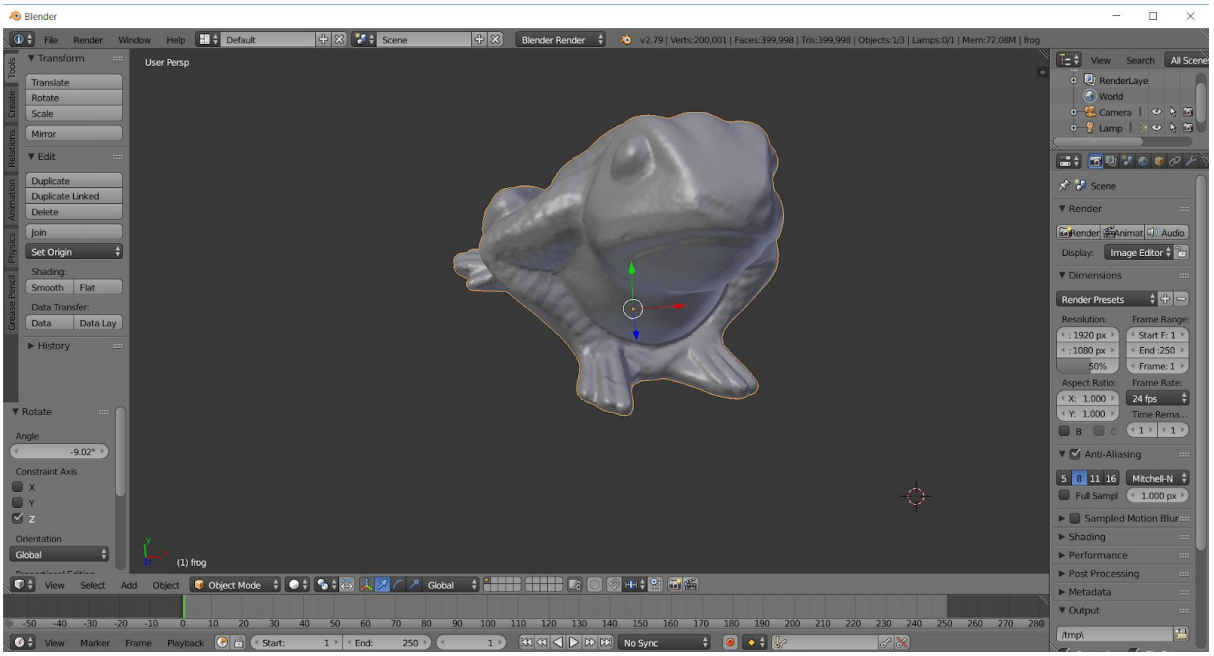


Figura 2.7. Interfície del programa Blender.

Actualment *Blender* és un programa gratuït de codi obert. Es troba en constant desenvolupament per part de la *Blender foundation* com a projecte de la comunitat i és un dels programes més complerts pel que fa a tractament de malles amb funcionalitats de mapeig de textures, disseny de malles o simulació de fluids[15].

MeshLab també és un programa gratuït de codi obert que treballa amb models de malles i ofereix eines per a simplificar i reparar malles, així com una gran quantitat de tècniques que s'utilitzen actualment en el processament de malles.

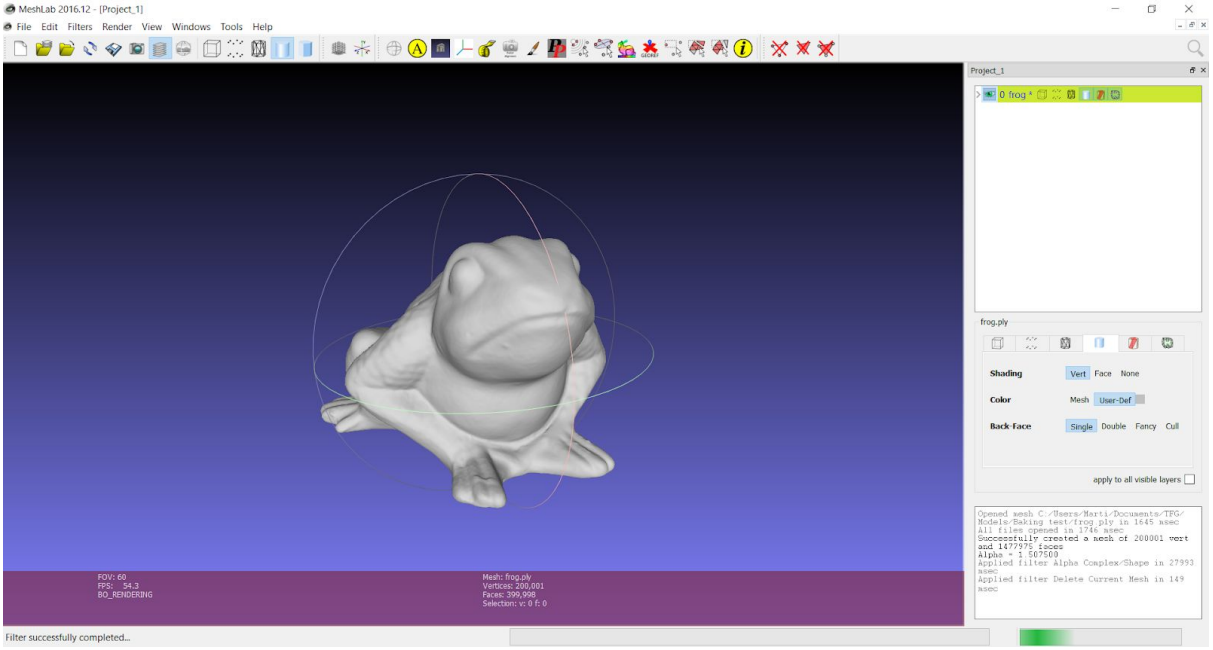


Figura 2.8. Interfície del programa MeshLab.

Per la seva banda, *Graphite* és una plataforma de recerca pel camp de gràfics desenvolupat per el projecte ALICE a INRIA Nancy Grand-Est / Loria, França[16]. Tot i ser gratuït i de codi obert, algunes de les seves funcionalitats requereixen de llicència per a poder utilitzar-les. Per a l'ús que en donarem en aquest projecte, la versió gratuïta ja ens serveix Ja que inclou tècniques de re-triangulació de malles i normalització de cares.

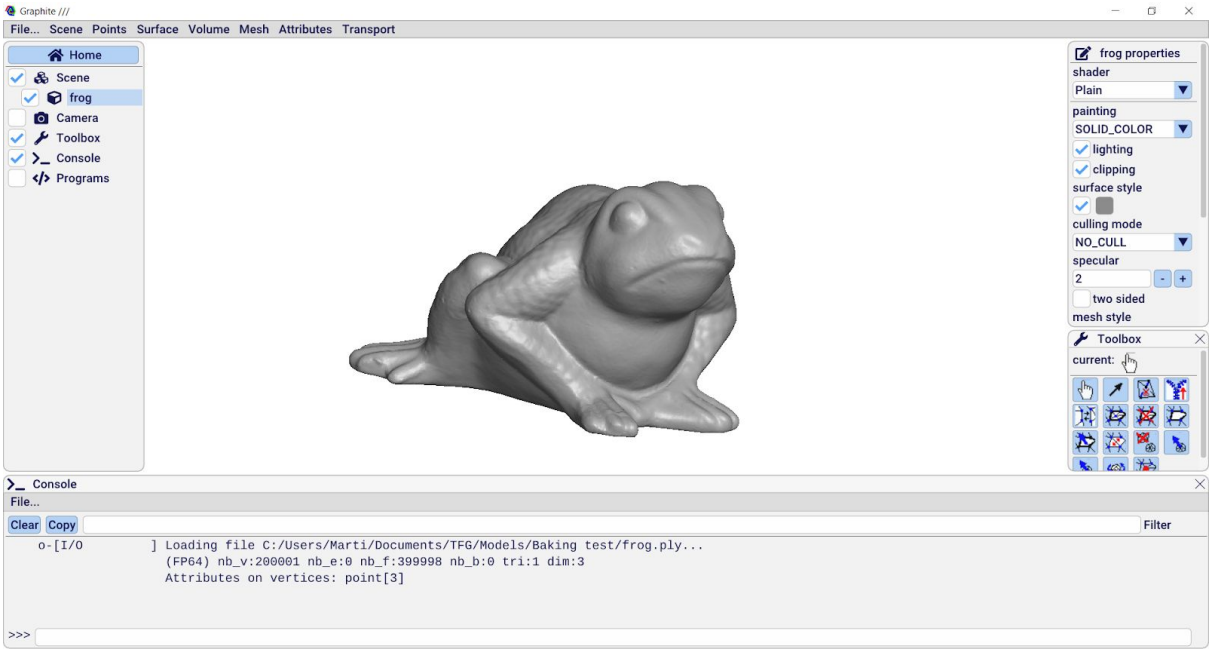


Figura 2.9. interfície del programa Graphite.

3. Estat de l'art

En aquest apartat parlarem de les tècniques actuals en el camp de la simplificació de malles triangulars, dels algorismes més utilitzats, les seves característiques i els seus operadors. així com de les tècniques de reconstrucció de superfícies.

3.1. Tipus d'algorismes de simplificació de malles

Pel que fal al camp de la simplificació de models de malles, és extens i divers el treball que s'hi ha fet prèviament, existint diversos algorismes que fan servir diferents tècniques per tractar aquest problema. Sense entrar en detalls de funcionament dels algorismes, podem dividir les simplificacions en dos tipus segons el paràmetre a optimitzar[1].

3.1.1. *Fidelity-Based Simplification*

Es tracta d'un tipus de simplificació en el que a l'entrada de l'algorisme es fa arribar un paràmetre de semblança, apart del model de malles a simplificar.

La semblança normalment es computa com la diferència entre la malla original i la malla simplificada (més endavant expliquem diverses formes de calcular-la), també s'anomena error ϵ.

Llavors es tracta de cercar el model de malla que satisfà la *semblança* i està format pel mínim nombre de triangles. Aquest problema és *NP-Hard* així doncs el que s'acaba generant és una simplificació amb un nombre petit de triangles (solució local) en comptes del nombre mínim de triangles (solució global)[1].

Aquest tipus de simplificació és adequat per a aplicacions on és més important la semblança de la simplificació que la interacció amb l'usuari.

3.1.2. *Budget-Based Simplification*

Es tracta d'un tipus de simplificació en el que a l'entrada de l'algorisme es fa arribar el nombre màxim de triangles, apart del model de malles a simplificar, llavors l'algorisme intenta minimitzar l'error ϵ de la simplificació.

Aquest tipus de simplificació es adequat per aplicacions on es prima la interacció amb l'usuaris o exiteix un límit de triangles per *frame*. Al no controlar-se l'error ϵ no es pot garantir cap nivell de semblança entre la malla original i la malla simplificada.

3.2. Mesures de la semblança

Com ja hem comentat, existeix el concepte d'error o semblança entre un parell de malles. Aquesta mesura ens permet controlar la deformació que pateix la malla quan es simplifica o la podem fer servir com a heurístic per a calcular un "cost" que ens permet escollir de forma òptima a cada pas de l'algorisme de simplificació.

Per a mesurar la semblança entre una simplificació i la malla original existeixen diferents tècniques. Les més importants són les següents.

3.2.1. Distància de Hausdorff

La distància de de Hausdorff es defineix com la distància entre parells de subsets d'un espai. S'utilitza per a mesurar l'error entre parells de tipus de dades tals com imatges, àudios i, en aquest cas, superfícies.

Formalment la distància de Hausdorff entre un punt *p* i una superfície *S* es defineix de la següent forma:

$$d(p, \mathcal{S}') = \min_{p' \in \mathcal{S}'} \|p - p'\|_2,$$

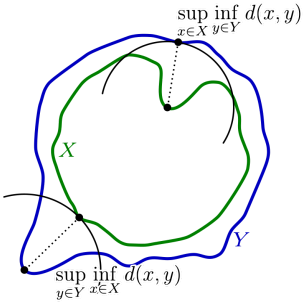
Figura 3.1. distància de Hausdorff entre un punt i una superfície [10].

I per tant la distància entre dues superfícies *S* i *S'*:

$$d(\mathcal{S}, \mathcal{S}') = \max_{p \in \mathcal{S}} d(p, \mathcal{S}').$$

Figura 3.2. distància de Hausdorff entre S i S' [10].

Cal tenir en compte però, que la distància entre dues superfícies no és simètrica per tant al final s'acaba definint com:



$$d_s(\mathcal{S}, \mathcal{S}') = \max\left[d(\mathcal{S}, \mathcal{S}'), d(\mathcal{S}', \mathcal{S})\right].$$

Figura 3.3. A l'esquerra: demostració de la no simetria de la distància de Hausdorff [8]. A la dreta: distància de Hausdorff entre dues superfícies [10].

3.2.2. *Quadric error*

El *quadric error* és una mesura de similitud de malles creada per Garland [2] i utilitza el concepte de pla de suport d'un vèrtex.

Un pla de suport d'un vèrtex està definit com el pla que conté una de les cares que està formada pel vèrtex en qüestió. Així doncs, un vèrtex té tants plans de suport com cares de les que forma part. Llavors el *quadric error* es defineix com la suma de les distàncies al quadrat amb aquests plans.

Si p = [a,b,c,d]^T representa al pla format per l'equació *ax+by+cz+d=0*, formalment es defineix per la següent fórmula:

$$\Delta(\mathbf{v}) = \Delta([v_x\ v_y\ v_z\ 1]^\mathsf{T}) = \sum_{\mathbf{p}\in\text{planes}(\mathbf{v})} (\mathbf{p}^\mathsf{T}\mathbf{v})^2$$

Figura 3.4. Fórmula de l'error d'un vèrtex [2].

La fórmula anterior es pot reformular en la seva forma quadràtica obtenint la següent expressió:

$$\begin{aligned}\Delta(\mathbf{v}) &= \sum_{\mathbf{p}\in\text{planes}(\mathbf{v})} (\mathbf{v}^\mathsf{T}\mathbf{p})(\mathbf{p}^\mathsf{T}\mathbf{v}) \\ &= \sum_{\mathbf{p}\in\text{planes}(\mathbf{v})} \mathbf{v}^\mathsf{T}(\mathbf{p}\mathbf{p}^\mathsf{T})\mathbf{v} \\ &= \mathbf{v}^\mathsf{T}\left(\sum_{\mathbf{p}\in\text{planes}(\mathbf{v})} \mathbf{K}_\mathbf{p}\right)\mathbf{v}\end{aligned} \qquad \mathbf{K}_\mathbf{p} = \mathbf{p}\mathbf{p}^\mathsf{T} = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix}$$

Figura 3.5. Fórmula del quadric error d'un vèrtex [2].

A la fórmula anterior K_p s'anomena l'error quadràtic fonamental i la seva suma té en consideració tots els plans de suport del vèrtex. Per tant, per a cada vèrtex v_i podem emmagatzemar el resultat de la suma a una matriu Q_i de mida 4x4.

3.3. Operadors de simplificació de malles

Entrant més en detall sobre com es realitzen les simplificacions, disposem de diversos operadors que ens permeten simplificar una malla localment. També podem classificar els algorismes segons l'operador que utilitzen. Alguns d'ells són:

- Vertex removal: consisteix en eliminar un vèrtex de la malla i re-triangular el forat que ha quedat a la malla. D'aquesta forma es redueix en un el nombre de vèrtex de la malla i en dos el nombre de triangles. Obtenint una malla més simple.

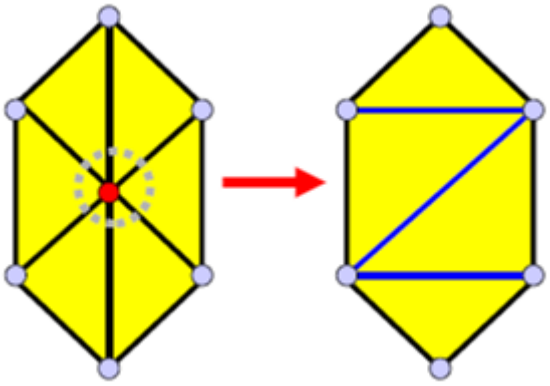


Figura 3.6. Exemple de vertex removal [3].

- Edge collapse/Half-Edge collapse: consisteix en eliminar una aresta de la malla i fusionar els dos vèrtex que la formàven en un de sol en algun punt de l'aresta suprimida. Quan el nou vèrtex està ubicat al mateix lloc que un dels que formaven l'aresta es tracta de un *Half-Edge collapse*, essent aquest un cas especial de l'*Edge collapse*[1][3]. De nou, reduïm el nombre de vèrtex en un i el nombre de triangles en dos. Obtenint així una malla més simple.

  És utilitzat en gran quantitat d'algorismes que es diferencien, en gran mesura, per com escullen l'aresta a suprimir. Aquests algorismes poden modificar la topologia del model de malles original però no poden unir regions inconnexes del model de malles.

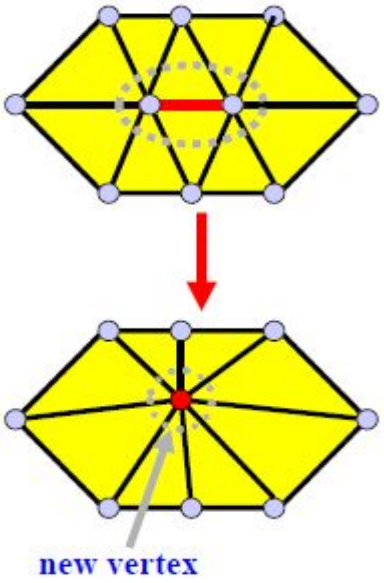


Figura 3.7. edge collapse [3].

- Vertex-Pair collapse o Virtual-Edge collapse: consisteix en fusionar dos vèrtex que no estiguin connectats en un nou vèrtex que està connectat a tots els vèrtex que estaven connectats als vèrtex originals i s'actualitzen d'acord a aquest canvi. Aquest operador pot afectar la topologia i normalment és limiten el nombre de parells de vèrtex candidats de forma heurística, ja que el nombre potencial de combinacions és $O(n^2)$ on n és el nombre de vèrtex de la malla[1].

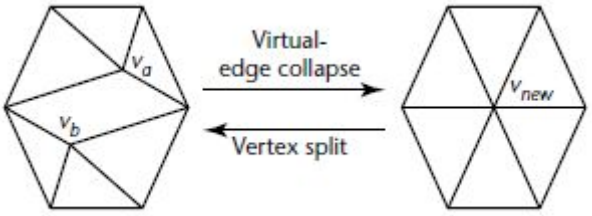


Figura 3.8. virtual-edge collapse [1].

Aquest operador és emprat per l'algorisme *Quadrics* o *Quadric error edge collapse*[2]. Algorisme que emprarem com a base per aquest projecte ja que al utilitzar aquest operador (que anomenen *aggregation*) treballa bé amb models de malles amb components no connexes entre elles tot i que no conserva la topologia del model de malles original.

- Triangle collapse: consisteix en fusionar els tres vèrtex d'un triangle en un nou vèrtex que està connectat a tots els vèrtex que estiguessin connectats els vèrtex originals. En el fons, és tracta de dos operacions de edge collapse concatenades.

- Cell collapse: consisteix en fusionar tots els vèrtex d'una partició de l'espai o pla (la cel·la) en un de sol. Aquest operador no preserva la topologia de la malla i el nivell de simplificació depèn de la mida de la partició. Utilitzat en *vertex clustering*, no permet un bon control de la malla simplificada resultant ja que depèn de forma indirecta de la mida de la partició[2].

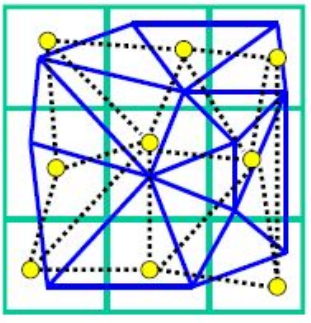


Figura 3.9. vertex clustering [3].

3.4. Algorismes de simplificació

Existeixen múltiples i variats algorismes de simplificació de models de malles. Nosaltres però, només ens centrarem en el *vertex clustering* i *quadric edge collapse decimation*. Ambdós algorismes molts utilitzats pels seus bons resultats o eficiència de càlcul.

3.4.1. *Vertex clustering*

L'algorisme de *vertex clustering* utilitza la idea de l'operador de cell collapse. Això el converteix en un algorisme de simplificació veloç i que retorna resultats correctes per a models amb topologies simples.

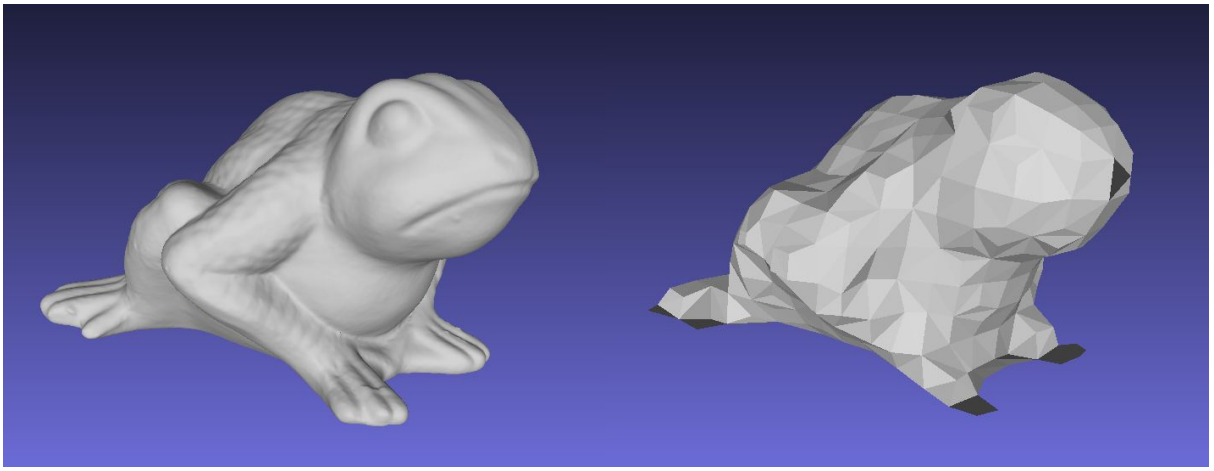


Figura 3.10. A l'esquerra: el model original. A la dreta: simplificació amb vertex clustering.

L'algorisme consisteix en crear una caixa contenidora del model d'entrada i generar-ne una graella segons la mida de la graella que s'ha especificat a l'entrada. Llavors, per a cada cel·la fusiona tots els vèrtex que aquesta conté en un de sol i crea una aresta amb tots els vèrtex que estaven connectats amb algun dels vèrtex fusionats.

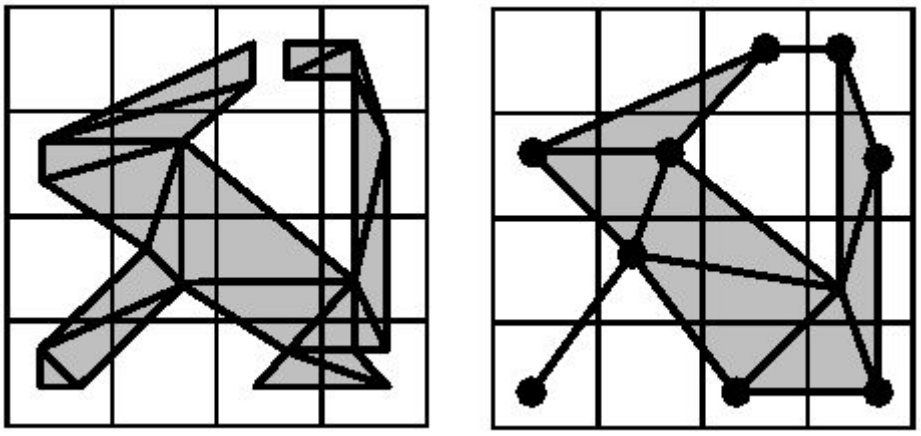


Figura 3.11. Exemple 2D de vertex clustering [9].

Aquest procediment permet a l'algorisme *vertex clustering* treballar amb qualsevol model d'entrada amb independència de la seva geometria i si és *manifold* o no. No obstant, el control sobre el model simplificat resultant és difícil, com ja hem esmentat a l'operador de *cell collapse*.

3.4.2. *Quadric error edge collapse*

L'altre algorisme que comentarem és el *quadric error edge collapse[2]* o com l'anomenem en alguns punts d'aquest document "*quadrics"*. Es tracta d'una algorisme més lent que el *vertex clustering* però que obté millors resultats. Com ja hem esmentat aquest algorisme fa servir l'operador de *virtual edge collapse*. Això li permet treballar amb models *non-manifolds*, tal com el *vertex clustering*, però també té un bon control sobre la qualitat del model simplificat resultant.

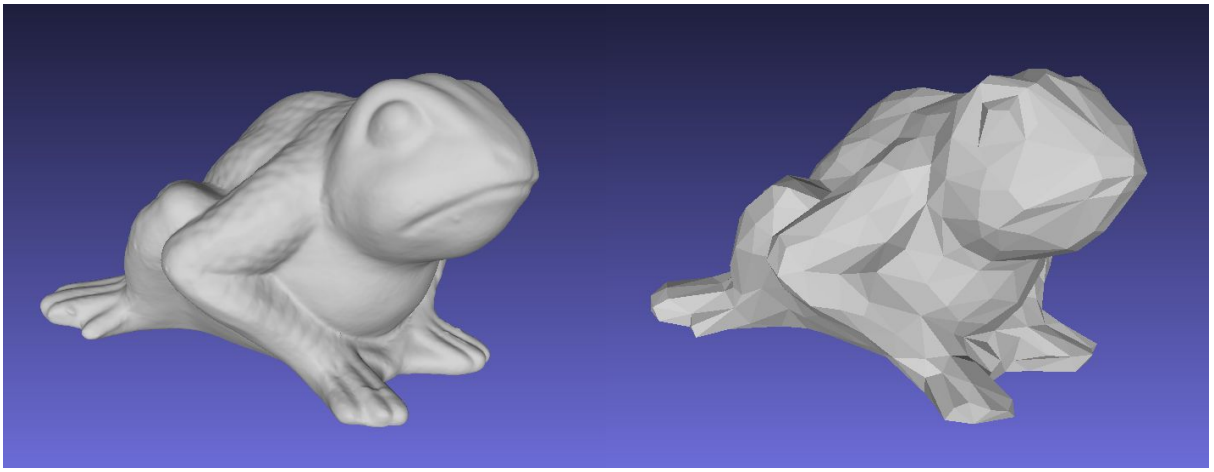


Figura 3.12. A l'esquerra model original. A la dreta: simplificació amb quadric edge collapse

L'algorisme utilitza el *quadric error* com a heuristic a l'hora de triar a quin parell de vèrtex aplica l'operador de simplificació. Triant a cada pas el parell de vèrtex que produeixen un error menor, seguint la següent fórmula[2]:

$$\Delta(V')=V'^{T}(Q_1+Q_2)V'$$

On Q_1 i Q_2 són les mesures de *quadric error* del parell de vèrtex als que es vol aplicar la operació i V' la posició del nou vèrtex resultant.

El conjunt de parells de vèrtex que als que es pot aplicar l'operador es determina a la inicialització de l'algorisme. Està format per tots els parells de vèrtex v_1 i v_2 connectats per una aresta $e = (v_1,v_2)$ i els parells de vèrtex u_1 i u_2 tals que la seva distància euclidiana és menor a un límit t.

En resum l'algorisme segueix els següents passos:

- Calcula la matriu Q_i per a cada vèrtex v_i.
- Selecciona tots els parells de vèrtex vàlids.

- Calcula la posició òptima del vèrtex resultant *v'* d'aplicar l'operador i el cost de l'operació.
- Crea una cua de prioritats segons el cost de l'operació.
- Realitza les operacions de forma iterativa sobre la cua. A cada pas s'actualitzen els costos de tots els parells afectats per l'operació.

3.5. Algorismes de reconstrucció de superfícies

La reconstrucció de superfícies és el problema que consisteix en, a partir d'un conjunt de punts a l'espai, generar o calcular la superfície que aquests punts descriuen. Hi ha diverses tècniques per a tractar aquest problema, les esmentarem per sobre i ens centrarem més concretament en la tècnica que hem utilitzat en aquest projecte.

Els algorismes de reconstrucció de superfícies existents poden ser d'interpolació o d'aproximació, segons si aquest treballa interpolant els punts d'entrada o generant una aproximació, i d'influència global o local, segons si els punts afecten sobre tota la superfície resultant o només a l'àrea pròxima al punt.

En el cas dels mètodes d'interpolació el conjunt de punts d'entrada també formen part de la superfície resultant. En canvi en el mètodes d'aproximació la superfície segueix la forma marcada pel conjunt de punts d'entrada però no hi passa exactament.

També els podem classificar en quatre grups segons la tècnica que utilitzen per a resoldre aquest problema: *sculpturing methods*, *volumetric methods*, *warping methods* i *incremental construction methods*.

Algorismes de *sculpturing*

Els mètodes de *sculpturing* són mètodes d'interpolació globals. Aquests mètodes utilitzen una triangulació inicial del conjunt de punts d'entrada per a decidir quins formen part de la superfície i quins no.

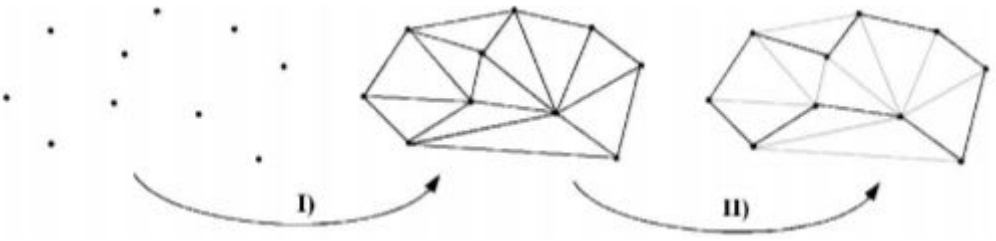


Figura 3.13. Idea general dels sculpting methods.
Primer (I) es crea una triangulació i després (II) es marca quins pertanyen a la superfície [12].

Normalment utilitzen la triangulació de Delaunay com a triangulació inicial. Això provoca una càrrega computacional important als algorismes, ja que en cas pitjor el càlcul de la triangulació de Delaunay és de $O(n^2)$ [12].

Algorismes volumètrics
Els mètodes volumètrics són mètodes d'aproximació que es basen de l'idea de convertir el conjunt de punts d'entrada en una representació volumètrica. Així doncs, primer creen el volum a partir del conjunt de punts d'entrada.

Per a crear la representació volumètrica es requereix certa funció de distància que estima la distància de cada punt a la superfície que s'està aproximant. Després la superfície es re-construeix utilitzant tècniques d'extracció d'i*sosurface*, pas que els converteix en algorismes d'aproximació.

Algorismes de *warping*
Els algorismes de *warping* són algorismes d'aproximació que requereixen una superfície aproximada del conjunt de punts d'entrada.

Per a poder funcionar correctament, aquesta aproximació inicial ha de ser tan propera al conjunt de punts d'entrada com sigui possible i ha de representar la topologia del conjunt de punts d'entrada. Durant l'execució de l'algorisme, aquesta superfície aproximada es va deformant.

A causa de que requereixen una superfície aproximada del conjunt de punts d'entrada solen utilitzar-se com a postprocessat d'altres algorismes.

Algorismes de construcció incremental
Aquests algorismes creen, de forma iterativa, la superfície a partir d'un element inicial. Aquest element inicial pot ser un vèrtex, una aresta o un triangle. Actuen de forma local a través del conjunt de punts d'entrada a mida que van reconstruint la superfície a base de cares triangulars.

Aquests algorismes no generen una aproximació de la superfície ja que treballen directament amb el conjunt de punts d'entrada per a reconstruir la superfície. Normalment requereixen que el conjunt d'entrada sigui uniforme per a poder donar bons resultats i requereixen estimacions de les normals a la superfície dels punts d'entrada.

No obstant, aquests algorismes tenen complexitat lineal en temps i espai [12].

3.5.1. *Alpha Shape*

L'*alpha shape* és un algorisme de reconstrucció de superfícies del tipus *sculpturing* que utilitza la triangulació de Delaunay com a triangulació inicial. L'algorisme, apart del conjunt de punts d'entrada, demana un valor $0 \leq alpha \leq +\infty$.

Si el valor d'*alpha* tendeix a zero, llavors el resultat acabarà sent el conjunt de punts d'entrada. En canvi, si el valor d'*alpha* tendeix a $+\infty$, el resultat serà la *convex hull* del conjunt de punts d'entrada.

Intuitivament podem explicar l'obtenció de l'alpha shape com una goma de radi *alpha* que no pot travessar el conjunt de punts d'entrada. Llavors podem dir que aquesta goma es va movent per l'espai eliminant tots els triangles que és capaç de travessar.

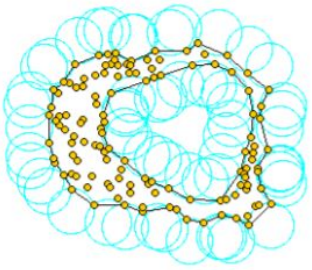


Figura 3.14. Exemple 2D d'alpha shape [12].

Podem veure que els valors d'*alpha* òptims depenen de les distàncies entre punts/vèrtex de cada model. Models amb distàncies entre vèrtex molt petites s'acosten a la *convex hull* amb valors d'*alpha* molt petits, com podem observar a la figura 3.15.

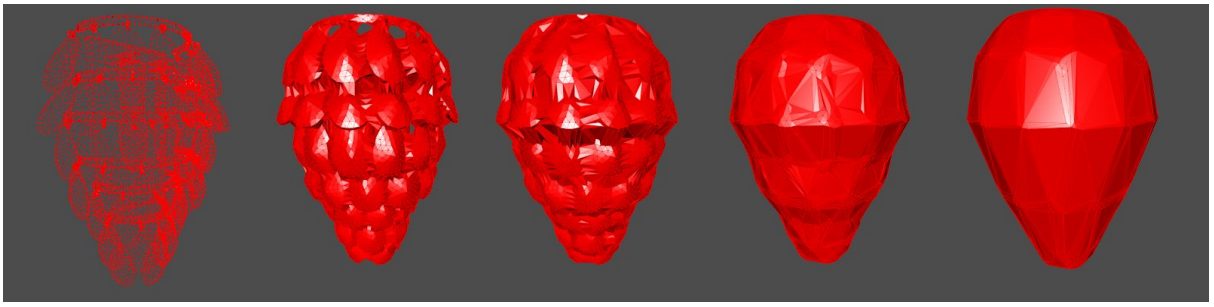


Figura 3.15. D'esquerra a dreta, alpha shape d'un model per valors d'alpha de 0.001, 0.005, 0.01, 0.05 i 0.5.

3.5.2. *Ball-pivoting*

L'algorisme de *ball-pivoting* és un algorisme de reconstrucció de superfícies del tipus construcció incremental. Funciona amb un principi similar a la goma d'esborrar de l'*alpha shape*, però en aquest cas diem que la goma té radi *p* i en comptes d'esborrar cares marca vèrtex.

L'algorisme comença amb la goma en contacte amb tres vèrtex, formant així el primer triangle de la superfície. A partir d'aquí la goma "pivota" sobre les arestes del triangle "tocant" així altres vèrtex. A mida que va tocant altres vèrtex els descarta si són interiors a la superfície i en cas contrari crea nous triangles.

4. Algorisme

L'algorisme que proposem consisteix en diverses fases diferenciades entre elles i es es van succeint, prenent com entrada la sortida del pas anterior, formant així el *pipeline* de l'algorisme.

L'idea general de l'algorisme consisteix en calcular una superfície envolupant del model d'entrada i aplicar un algorisme de simplificació sobre aquesta superfície envolupant en comptes de sobre el model original.

A la figura 4.1. podem veure el diagrama de flux de l'algorisme amb tot els passos dividits en preprocessat, cos de l'algorisme i postprocessat. Apart de les entrades i sortides de cada pas, podem observar passos opcionals marcats pel contorn discontínu de la caixa del seu pas.

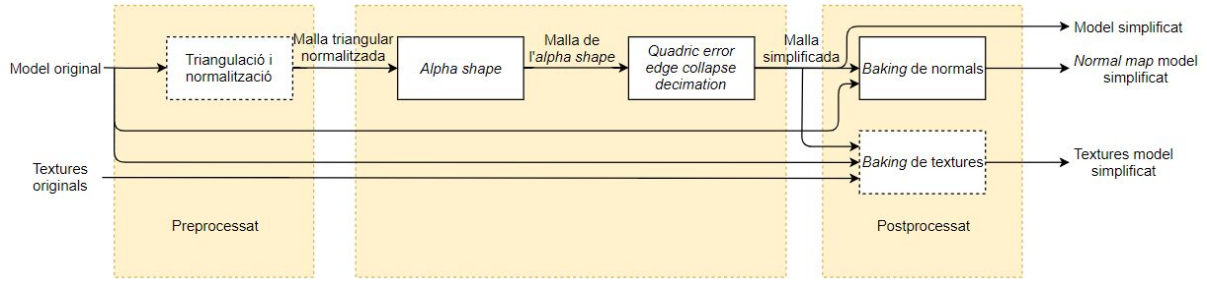


Figura 4.1. Diagrama de flux de l'algorisme.

Un tipus de superfície envolupant d'un model és la *convex hull*, com ja hem vist, aquesta pot tenir molt poca o cap semblança amb el model original. Per tant, optarem per l'algorisme d'*alpha shape*. Aquest algorisme ens permet, segons el valor del paràmetre *alpha*, controlar com s'ajusta la superfície envolupant al model original.

Per a poder aplicar l'*alpha shape* al model, necessitem realitzar un preprocessament per assegurar que la malla pot ser tractada amb expectatives de bons resultats. En aquest preprocessament, convertirem el model d'entrada en una malla triangular en cas de no ser-ho i normalitzem la mida de totes les cares del model i així assegurem que l'entrada de l'*alpha shape* serà uniforme. Com veurem més endavant, aquest preprocessament pot arribar a ser opcional.

Treballar sobre la superfície envolupant i no sobre el model original evita o retarda en gran mesura que es perdi geometria al realitzar la simplificació, cosa que a tots els algorismes acaba passant quan el nivell de simplificació del model és prou gran. Com a contrapartida, perdem bona part o tota la topologia del model original.

Així doncs, en el pas de simplificació del model, l'algorisme no rebrà el model original com entrada, sinó aquesta superfície envolupant. Per tant, el model d'entrada a l'algorisme ja arriba amb un cert error respecte el model original. Per això hem obtat per utilitzar

l'algorisme de simplificació de *Quadric error edge collapse,* ja que obté simplificacions més similars als model d'entrada que altres algorisme com el *vertex clustering*.

Finalment, afegim un pas de postprocessament a l'algorisme. Aquest pas computa, a partir de la malla resultant simplificada i la malla original, el *normal map* de la malla simplificada. Apart, si la malla original tingués textures, també obté el mapeig de la textura pel model simplificat.

Amb aquest postprocessat la malla resultant pot simular la topologia del model original i obtenim resultats molt més convincents.

5. Implementació

5.1. Preprocessament: normalització de cares

En el preprocessament el que busquem és adecuar la malla d'entrada per a que pugui ser processada per l'algorisme garantint les condicions d'entrada. Aquestes condicions són tenir una malla triangular i que les mides de les cares estiguin normalitzades.

Aquest pas és important per assegurar que durant el següent pas, de l'*alpha shape*, no es generen forats a la malla a causa d'utilitzar valors d'*alpha* massa petits per a certes cares, ja que els algorismes de reconstrucció de superfícies requereixen un bon mostreig de les dades d'entrada.

Per a realitzar aquest preprocessament hem obtat per utilitzar el programa *Graphite*. Aquest programa inclou les funcionalitats que necessitem per a realitzar aquest pas.

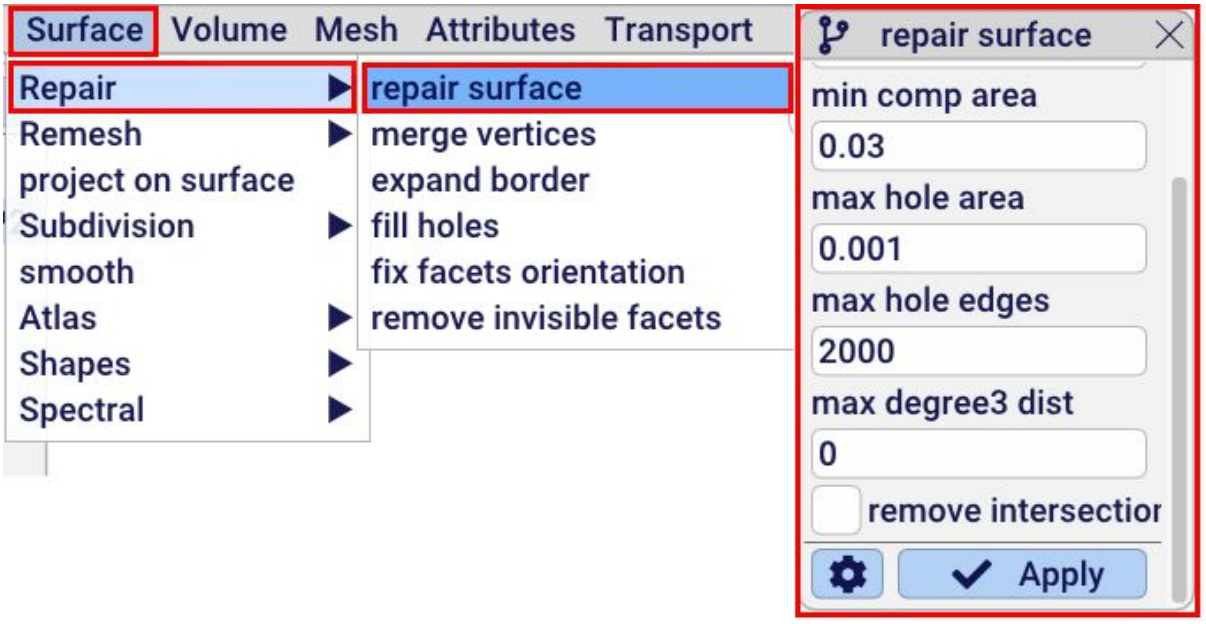


Figura 4.1. Ús del repair de Graphite.

Per a poder normalitzar les cares del model d'entrada sovint haurem d'utilitzar l'opció de repair del mateix *Graphite* ja que la malla d'entrada a la normalització ha de complir certes certes condicions.

Aquesta opció també serveix per a convertir models de malles no triangular a malles triangulars.

Ara ens centrarem en les opcions que tenim quan volem reparar una malla. D'entre les opcions que disposem la que ens interessa és el "min comp area" o "àrea mínima del component". Si no s'especifica correctament, aquest paràmetre pot comportar una deformació del model significativa i en alguns casos la pèrdua de geometria.

El paràmetre "min comp area" és un valor entre 0 i 1 que defineix el percentatge mínim d'àrea que ha de tenir una zona connexa del model sobre l'àrea total del model per a que no sigui descartada pel repair. On el 0 significa que no s'eliminarà geometria i l'1 que molt probablement obtindrem un model buit com a resultat, si el model no és tot ell connex.

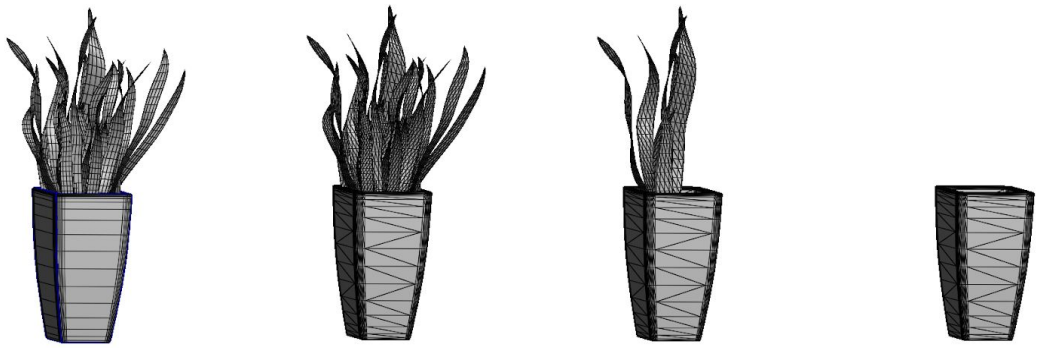


Figura 4.2. D'esquerra a dreta. Model original format per quads. Resultat d'aplicar repair a Graphite amb valors de "min comp area" de 0, 0'03 i 0'06.

On cop s'ha reparat el model d'entrada ja podem normalitzar la mida de les seves cares. Quan volem normalitzar la mida de les cares d'un model utilitzem l'opció "*remesh smooth*" de *Graphite* que refà la malla del model a partir de l'algorisme *Lp Centroidal Voronoi Tessellation* [20]. Aquesta operació ens deixa diversos paràmetres per poder aconseguir el model normalitzat que desitgem.

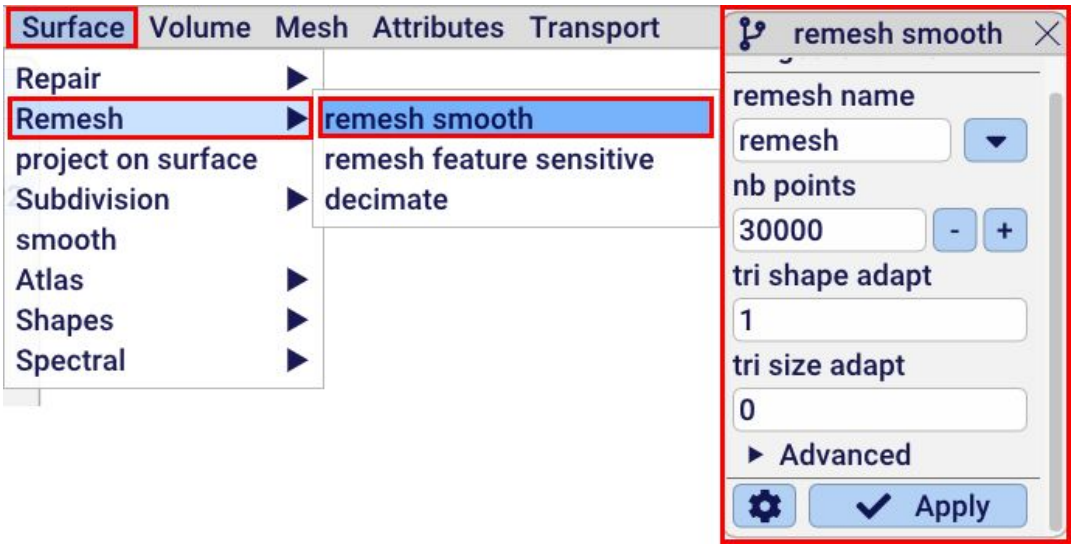


Figura 4.3. Normalització de les cares amb el remesh de Graphite.

Els paràmetres en els que ens centrarem són el “nb points” o nombre de punts, “tri shape adapt” o variació de la forma del triangle i “tri size adapt” o variació de la mida del triangle. Tots tres estan enfocats a donar flexibilitat a l’operació.

El primer paràmetre, “nb points”, ens permet especificar quants vèrtex volem que tingui el model normalitzat aproximadament. Donar un valor adequat a aquest paràmetre és important, ja que si fixem un objectiu amb massa pocs punts es produirà una pèrdua de geometria a la malla normalitzada respecte la malla d’entrada. En canvi, si especifiquem massa punts, la mida dels triangles cada cop serà més petita, provocant un sobrecost a tot l’algorisme.

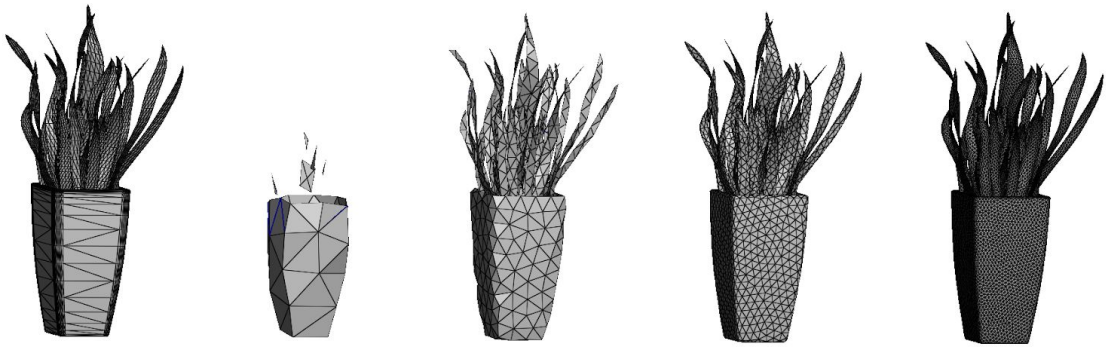


Figura 4.4. D’esquerra a dreta. Model reparat seguit de models normalitzats amb valors de “nb points” de 100, 1000, 5000 i 30000.

Els paràmetres “tri shape adapt” i “tri size adapt” són dos valors entre 0 i 1. El 0 indica rigidesa total a l’hora de crear els triangles pel que fa la forma i mida respectivament i l’1 indica certa flexibilitat a l’hora de donar forma i mida als triangles. Aquest paràmetres no són tant crítics com el “nb points” però ens poden ajudar a obtenir malles normalitzades amb menys triangles. Això sí, augmentant el risc de produir forats a la malla en el pas d’*alpha shape*.

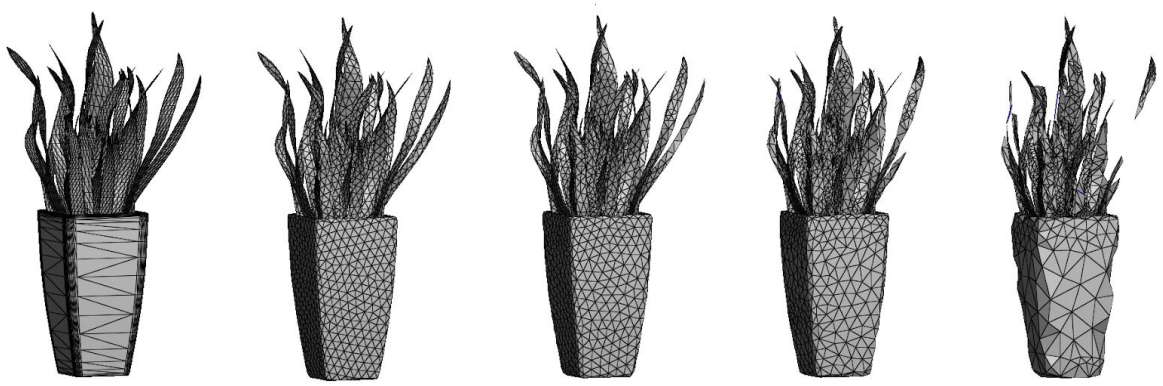


Figura 4.5. D'esquerra a dreta. Model reparat seguit de models normalitzats amb valors de "tri size adapt" de 0, 0'3, 0'6 i 1 i nombre de punts objectiu constant.

Com podem veure a la figura 4.5 fer servir valors de "*tri size adapt*" massa alts provoca deformacions i pèrdua de geometria a la malla resultant. Si ho combinem amb el fet que en aquest pas volen normalitzar la mida de les cares de la malla, optarem per utilitzar valors de zero o molt propers a zero per aquest paràmetre

5.2. Pas d'*alpha shape*

Un cop realitzat el preprocessat del model d'entrada i obtingut el model normalitzat ja podem aplicar el pas de l'*alpha shape*. Aquest pas consisteix en aconseguir una superfície envolupant del model que no suposi tanta pèrdua de detall com la *convex hull* d'un model. Per a fer-ho apliquem la tècnica de reconstrucció de superfícies que dona nom a aquest pas i que ja hem explicat amb anterioritat.

Hem implementat un script en R que utilitza la llibreria de R *alphashape3D*[5]. Donat un model normalitzat pel *Graphite* i un valor de alpha aquest script ens retorna el seu *alpha shape*. Posteriorment podem escriure el resultat en un fitxer en format .stl ASCII.

L'*alpha shape* que obtenim com a resultat té diversos camps. Els que ens interessen a nosaltres són, la taula que conté la informació dels triangles que s'han generat i el vector de punts. La taula dels triangles indica amb un codi numèric del 0 al 3, si el triangle no pertany a l'*alpha shape*, si és interior a l'*alpha shape*, si és regular o singular respectivament i els índex dels tres vèrtex que el formen.

Per tant, amb aquesta informació podem escriure fàcilment en un fitxer de sortida en format .stl totes les cares que pertanyen a l'*alpha shape*. És a dir, les marcades com a regulars i singulars.

5.3. Simplificació amb *quadrics*

Per a realitzar el pas de simplificació del model de l'*alpha shape* hem obtat per a fer servir el programa *MeshLab*. Aquest programa conté les implementacions dels algorismes de simplificació *vertex clustering* i *quadric error edge collapse*. També és capaç de llegir fitxers

en format .stl, per tant no suposarà un problema el format del fitxer de sortida del pas anterior.

Un dels problemes que sí que ens trobem, és que les cares del model de l'*alpha shape* no estan orientades de forma consistent, provocant una alternança en l'orientació de les cares del model.

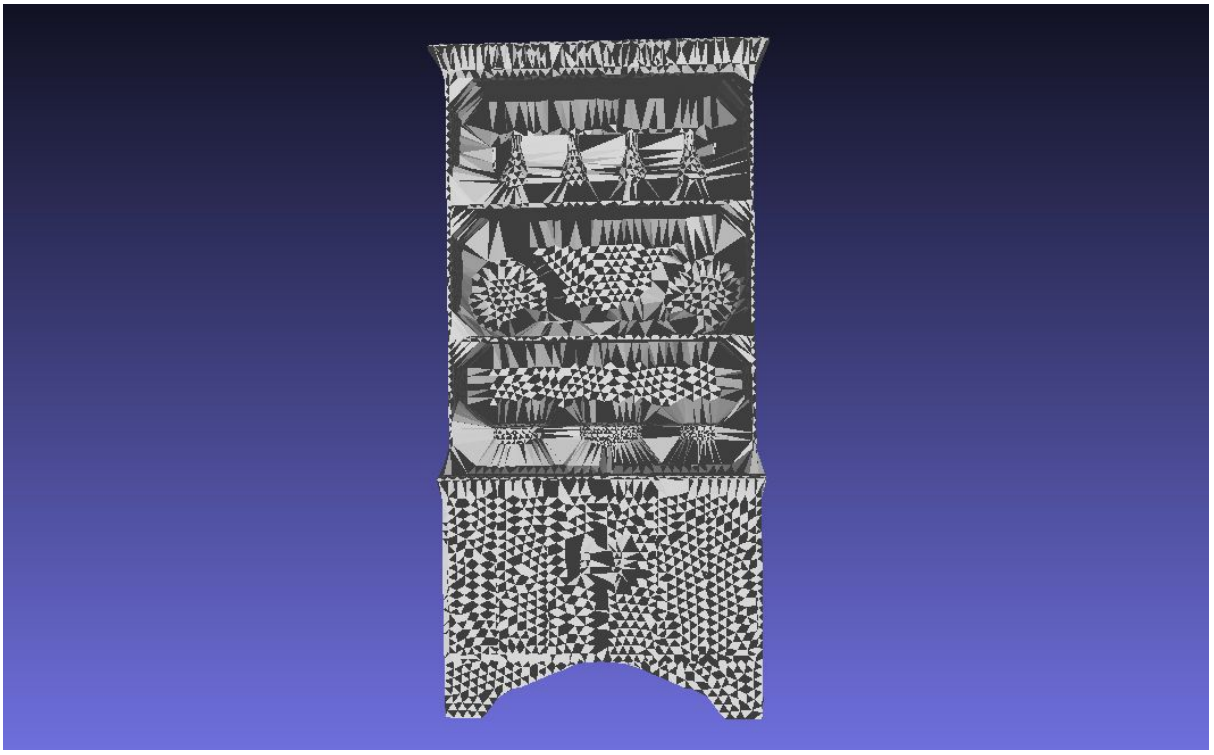


Figura 4.6. Model de l'alpha shape amb les cares orientades de forma incoherent.

MeshLab inclou la funcionalitat de reorientar les cares de forma coherent que podem utilitzar sense generar massa sobrecost computacional. En cas de que el resultat de reorientar les cares ens deixi les cares interiors a l'exterior de la malla només les hem d'invertir.

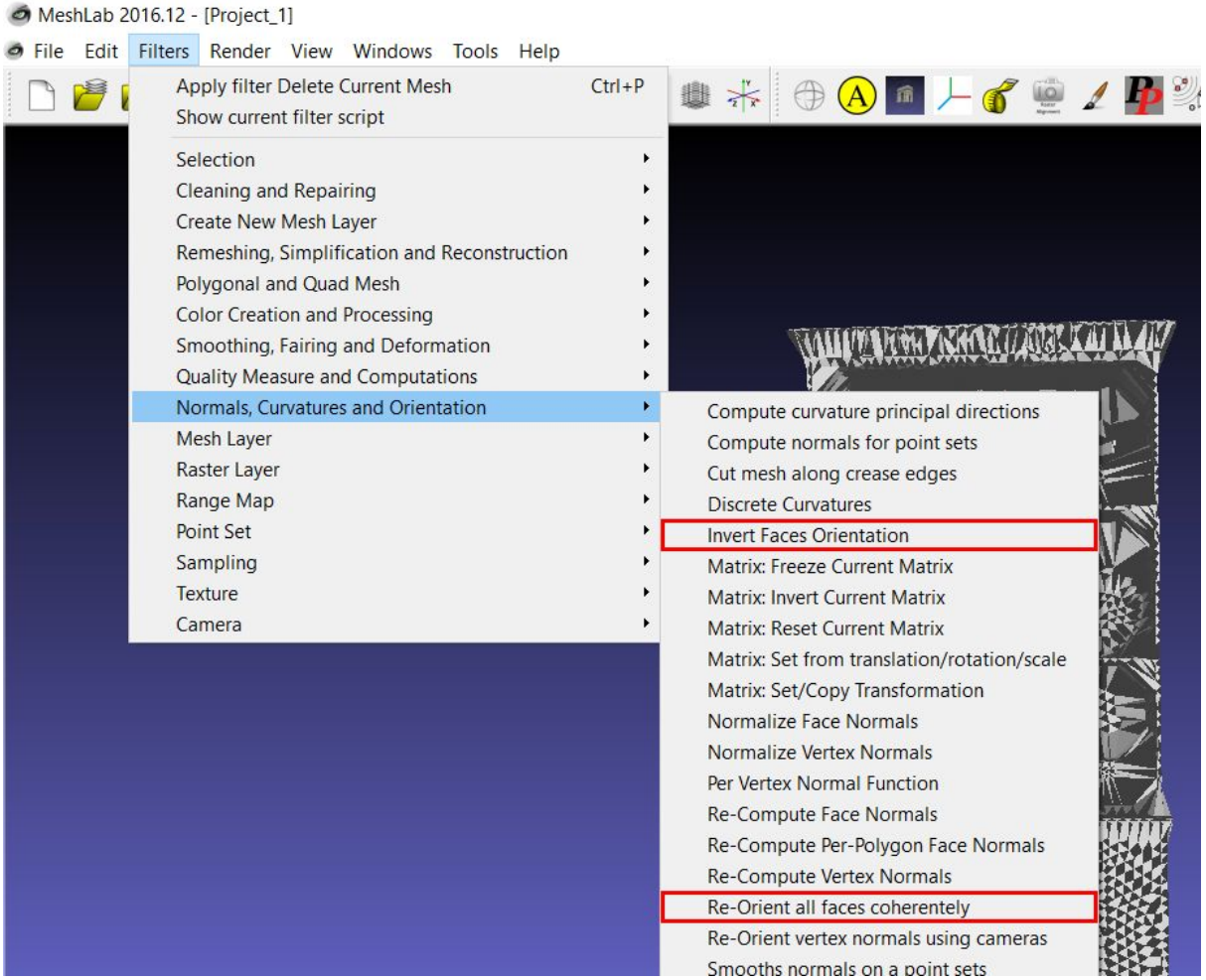


Figura 4.7. Tècniques de reorientació de cares a MeshLab.

Un cop tenim les cares orientades de forma coherent generem les simplificacions amb el nombre de cares objectiu. Abans d'executar el pas ens fixem en diversos paràmetres que podem tocar. Aquests els podem dividir en dos tipus segons si és un paràmetre o una opció de l'algorisme.

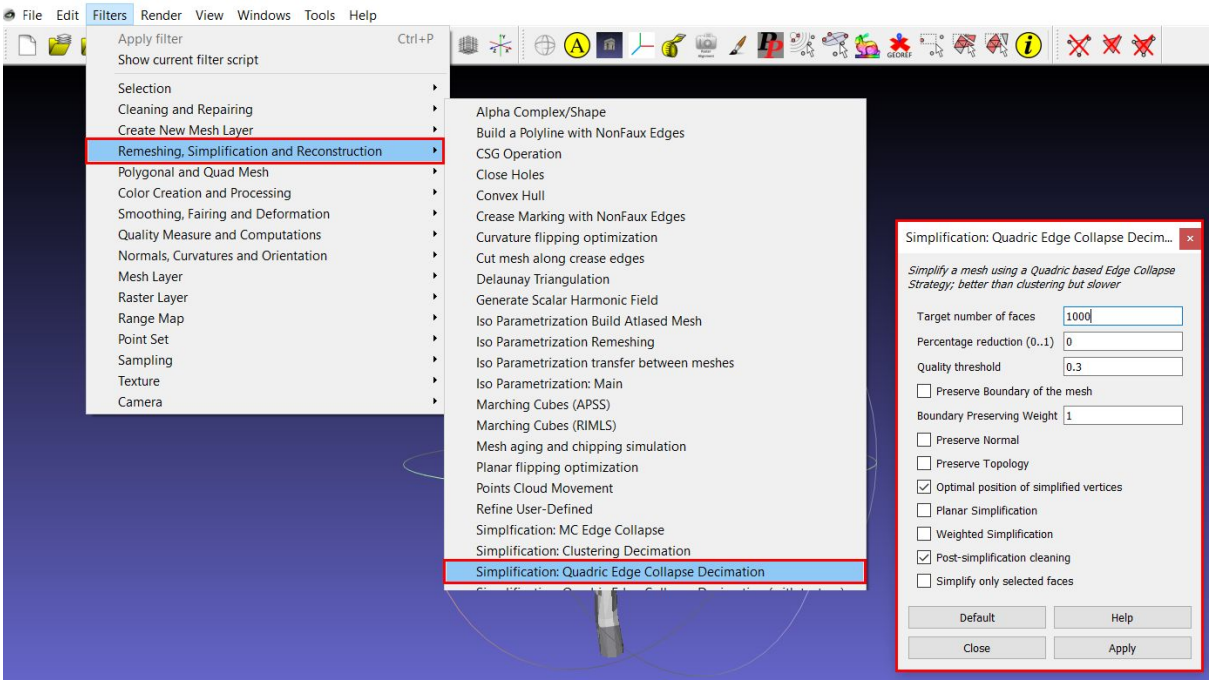


Figura 4.8. Quadric error edge collapse a MeshLab.

Entre els paràmetres que podem definir trobem el ”*target number of faces*”, “*percentage reduction*”, “*quality threshold*” i “*boundary preserving weight*”.

El paràmetre “*target number of faces*” serveix per a definir el nombre de cares que volem que tingui la malla simplificada resultant. Ha de ser un nombre enter més gran que zero i menor que el nombre de cares de la malla d’entrada.

El “*percentage reduction*” és similar al “*target number of faces*”. És un valor entre zero i u que indica en quin percentatge es vol reduir la mida de la malla d’entrada a través de la simplificació. En cas de ser zero aquest paràmetre no es té en compte. Si s’especifiquen el nombre de cares objectiu i el percentatge de reducció alhora s’utilitza el que comporta una major simplificació al model de sortida.

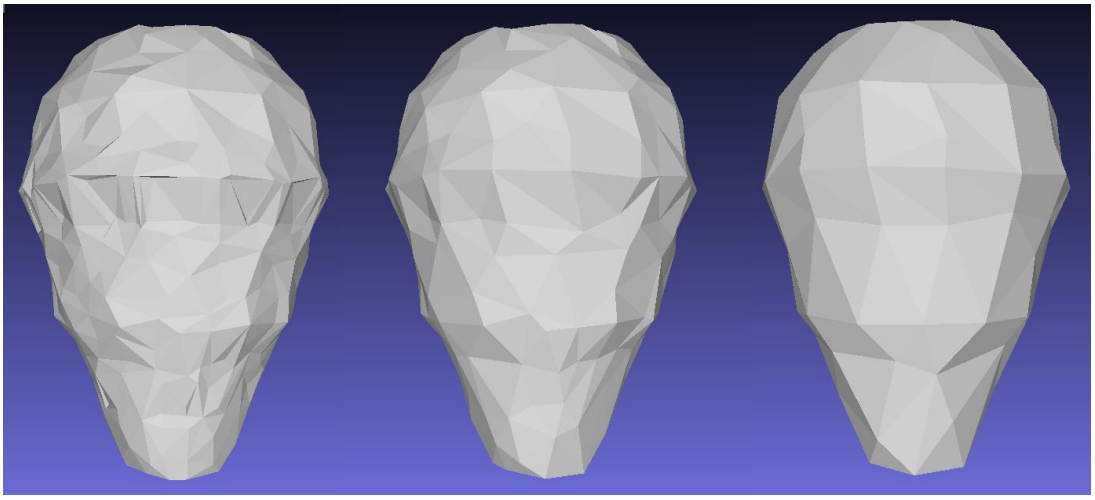


Figura 4.9. D’esquerra a dreta. Models simplificats amb valors de nombre de cares objectiu de 1000, 500 i 250.

El “*quality threshold*” és un valor entre zero i u que serveix per a indicar fins a quin punt es considera bona una cara a l’hora d’afegir-la a la simplificació. Un valor de zero indica que s’accepten totes les cares. Per valors més grans, penalitzen les cares que no s’adapten a la forma del model d’entrada.

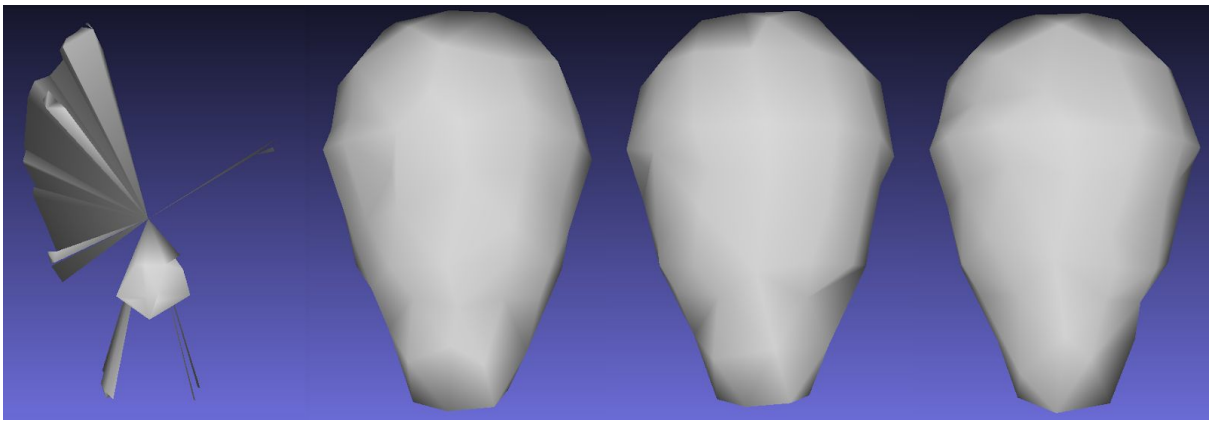


Figura 4.10. D’esquerra a dreta. Models simplificats amb valors de quality threshold de 0, 0’05, 1 i 2.

Podem veure que acceptant qualsevol cara a la simplificació obtenim un model que no té cap semblança amb l’entrada però augmentant poc aquest valor les simplificacions ja no són tant desastroses. Si el valor segueix augmentant la qualitat de les simplificacions tampoc varia molt per tant hem utilitzat l’algorisme amb el valor per defecte de 0’3.

El “*boundary preserving weight*” és un valor enter positiu. Indica com d’important és mantenir els vèrtex de la superfície del model. Com que el nostre model d’entrada és una superfície i per tant tots els vèrtex pertanyen a aquesta superfície, el valor d’aquest paràmetre no és rellevant per al nostre problema.

Pel que fa a opcions de l’algorisme trobem les següents:

“*Preserve boundary of the mesh*”: si es marca aquesta opció l’algorisme intenta no tocar els vèrtex de la superfície. Igual que en el cas del “*boundary preserving weight*” no té sentit que el fem servir ja que la malla d’entrada és una superfície.

“*Preserve normal*”: quan es marca aquesta opció l’algorisme intenta mantenir la orientació de les cares de la superfície. Si tenim en compte que en el past de postprocessat realitzem el *normal map* del model original a al model simplificat aquesta opció no ens és necessària.

“*Preserve topology*”: aquesta opció serveix per indicar a l’algorisme que només pot aplicar l’operador de simplificació als parells de vèrtex que tenen una aresta entre ells i així mantenir la topologia. Com que ja partim de que hem perdut bona part de la topologia en el pas de l’*alpha shape* i l’algorisme que proposem no pretén mantenir-la no utilitzarem aquesta opció.

"*Optimal position of simplified vertices*": Aquesta opció serveix per a marca a l'algorisme si volem que utilitzi l'operador de *edge collapse* triant la posició del nou vèrtex que redueix la mesura error quadràtic o l'operador de *half-edge collapse*. En el nostre cas volem utilitzar l'operador de *edge collapse* per a reduir tant com es pugui l'error a la simplificació, per tant l'activem.

"*Planar simplification*": Aquesta opció serveix per a millorar la simplificació de les àrees planes de la malla d'entrada. Com que la malla que arriba a el pas de simplificació no és el model original sinó la seva *alpha shape*, no utilitzarem aquesta opció. Ja que les parts planes de l'*alpha shape* potser no ho eren en el model original.

"*Weighted simplification*": Quan es marca aquesta opció, a cada vèrtex s'assigna un pes segons la seva qualitat. Aquest pes actua de factor multiplicatiu a la mesura d'error quadrat. Llavors l'algorisme actua a les zones amb menys qualitat de forma molt concentrada. L'algorisme que proposem no té com objectiu actuar en zones concretes de la malla ja que de fet aquesta arriba en forma de superfície a aquest pas. Per tant, no utilitzarem aquesta opció.

"*Post-simplification cleaning*": Si es marca aquesta opció, l'algorisme realitza un seguit d'operacions de post processat per a netejar la malla simplificada resultant. Per defecte, activarem aquesta opció.

5.4. Postprocessament: *Baking*

Un cop tenim la malla simplificada ens disposem a "transferir" les propietats del model original al model simplificat resultant. Ens centrem en transferir els valors dels vectors normals de la superfície del model original al model simplificat i les textures, si el model original en té.

Per a fer-ho utilitzem la tècnica de *baking* a través del programa *Blender*. Per a poder fer el *bake* de les normals, requerim el model original i el model simplificat carregats a *Blender*. Els dos models han d'estar orientats i posicionats igual i han de tenir les mateixes proporcions. També hem de comprovar que el renderitzat de models és suau. De no ser-ho les arestes quedaran marcades al normal map resultant.

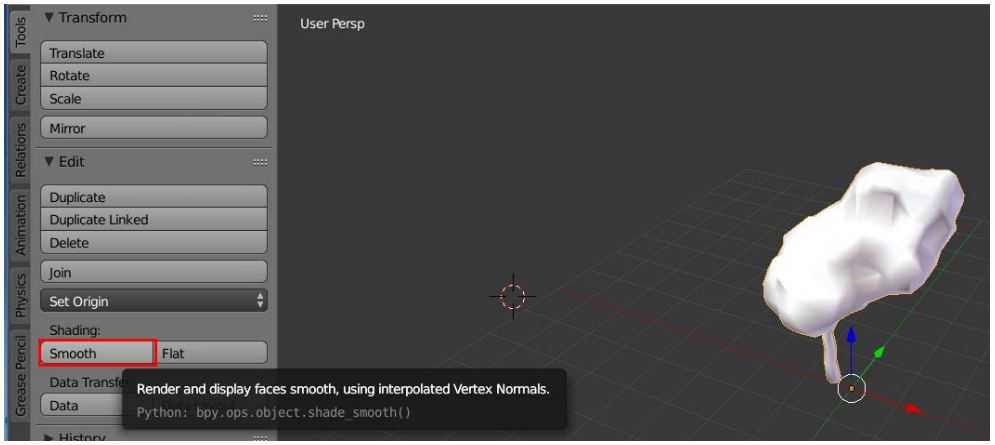


Figura 5.11. Renderitzat de models suau a Blender.

No obstant, en aquests primers passos mantindrem ocult el model original ja que només treballarem amb el model simplificat.

El primer que hem de fer és desplegar el model simplificat per obtenir les seves coordenades de textura i per a poder-li assignar una textura buida. Aquestes funcionalitats es troben a la interfície *UV Editing* de *Blender*.

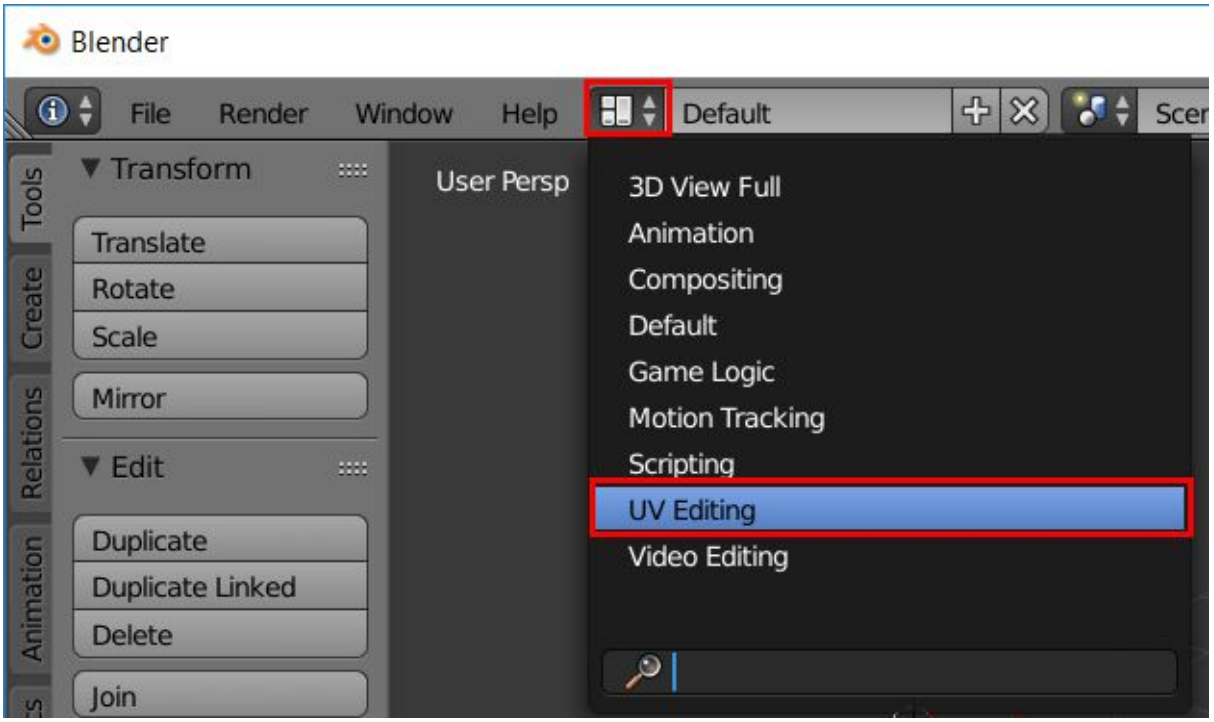


Figura 5.12. Accés a la interfície de UV Editing a Blender.

Per a poder desplegar un model de malles l'hem de tenir seleccionat en el *Edit Mode*, sinó les opcions de desplegat no estan disponibles.

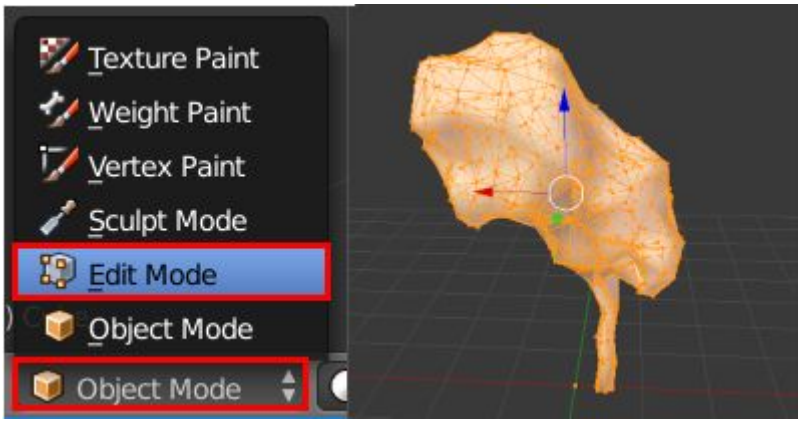


Figura 5.13. A l'esquerra canvi a Edit mode. A la dreta: Model seleccionat amb Edit Mode.

Un cop preparat l'entorn per a desplegar el model, utilitzarem l'opció *Smart UV project* al menú de *Unwrap* que podem accedir fàcilment amb l'accés ràpid a la tecla "u". Aquesta

tècnica de projecció mapeja tota la superfície del model dintre de la textura assignada amb resultats correctes per a la majoria de models. Aquest desplegat del model no és coherent amb les adjacències ni ordre de les cares.

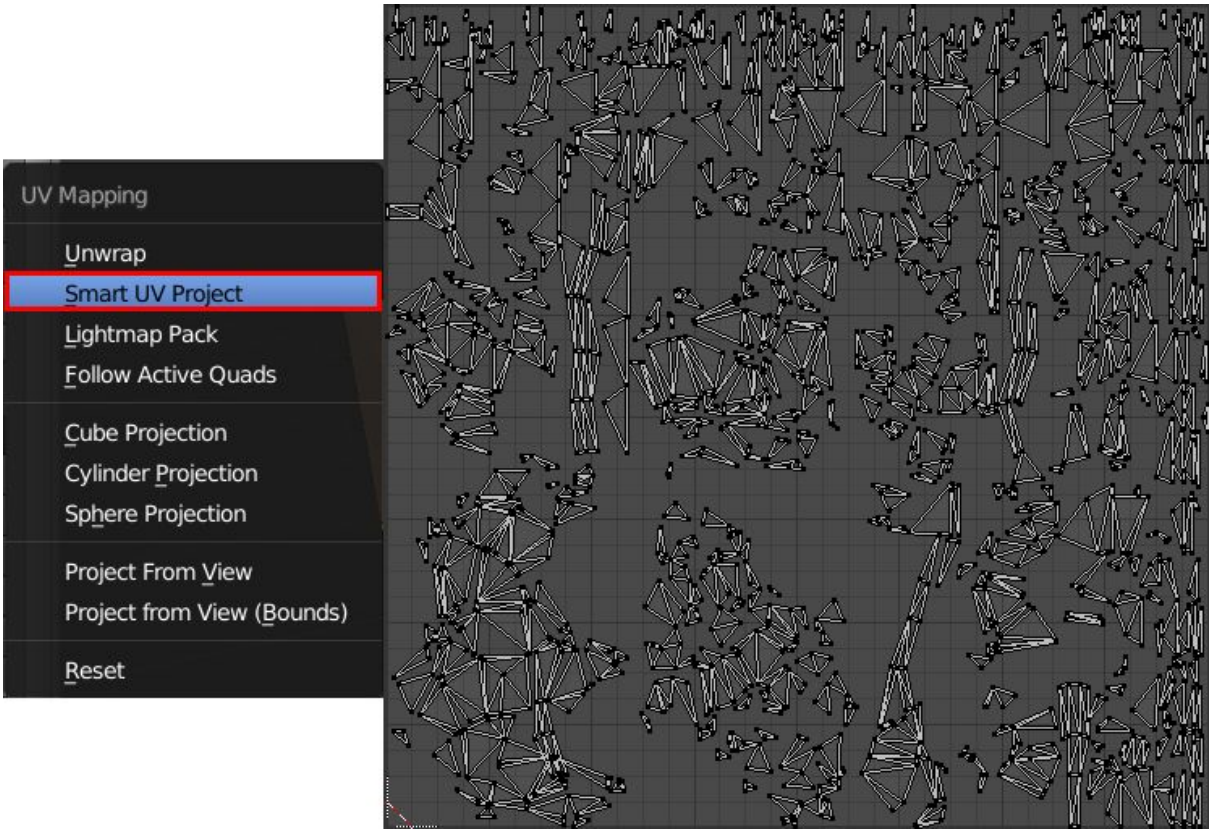


Figura 5.14. A l'esquerra: opcions de desplegat a Blender. A la dreta: exemple de model desplegat amb l'opció Smart UV Project.

Quan tenim el model desplegat hem d'assigar-li una textura per a que s'hi recullin els resultats del *bake*. Podem carregar un textura o crear-ne una de nova a *Blender*. En aquest cas que estem creant les textures i *normal maps* dels models simplificats crearem noves textures des de *Blender*.

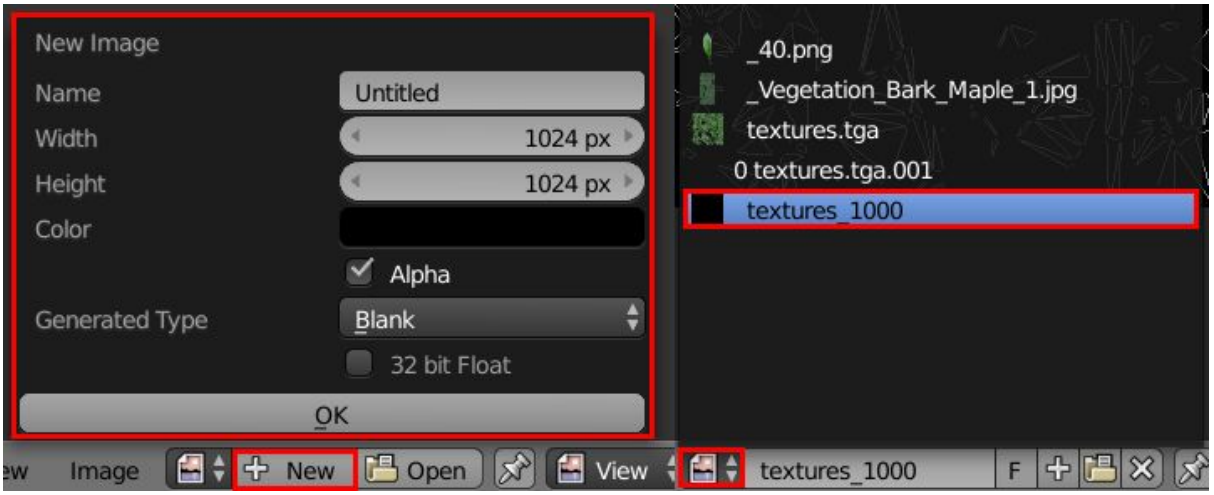


Figura 5.15. A l'esquerra creció d'una nova textura. A la dreta: Associació d'una textura a un model.

Un cop hem desplegat el model simplificat i li hem assignat textura podem tornar a la interfície per defecte de *Blender* i revelar el model original, llavors seleccionem el model original primer i després el model simplificat (shift+*LMB*).

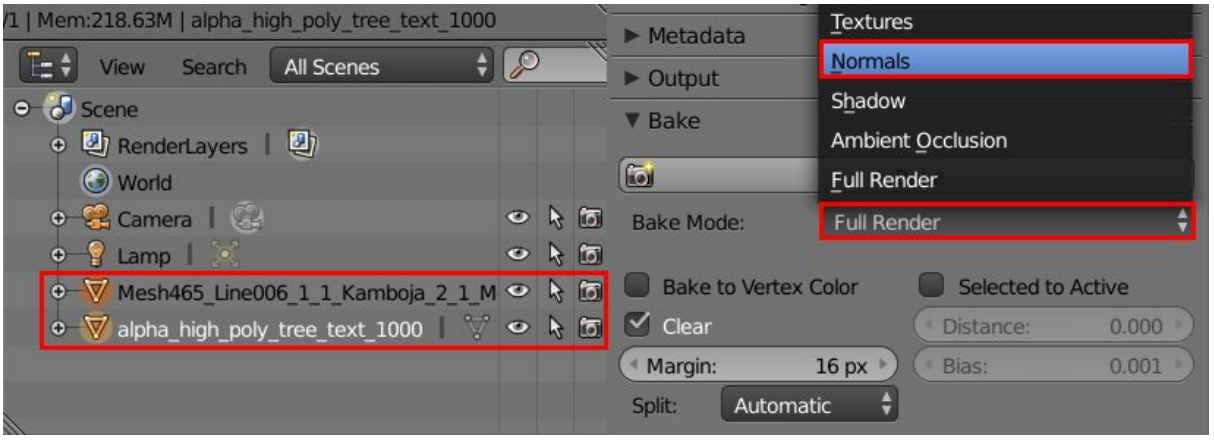


Figura 5.16. A l'esquerra: model simplificat actiu (alpha) i model original seleccionat (Mesh465). A la dreta: activar opció de bake de normals.

Podem veure que tenim diverses opcions generals a l'hora de fer *bake.* La primera opció, "*Bake to Vertex Color*" és per a que en comptes d'escriure els resultats a la textura assignada al model, s'escriguin a la propietat vertex color dels vèrtex del model simplificat. Nosaltres volem que els resultats s'escriguin a la textura per tant, no activarem aquesta opció.

L'opció "*Selected to Active*" serveix per indicar que volem fer bake de les propietats del model seleccionat al model actiu. Aquesta opció serveix per a transferir propietats entre models per tant, sempre la tindrem activada. L'opció "*Clear*" indica que s'esborri qualsevol contingut que tingués la textura objectiu abans de fer el *bake*. Per evitar errors derivats de sobreescriure textures també la deixarem activada. En aquest punt ja podem fer el *bake* de les normals del model original al model simplificat i veiem que s'activa el paràmetre "*Distance*" o distància.

El valor del paràmetre de distància serveix per a limitar quines parts del model original es tenen en compte per a fer el *bake*. Si la distància que separa una part del model original al model simplificat és més gran que la que s'indica en aquest paràmetre, no es té en compte. Si es deixa el valor per defecte de zero, aquest paràmetre no té afecte.

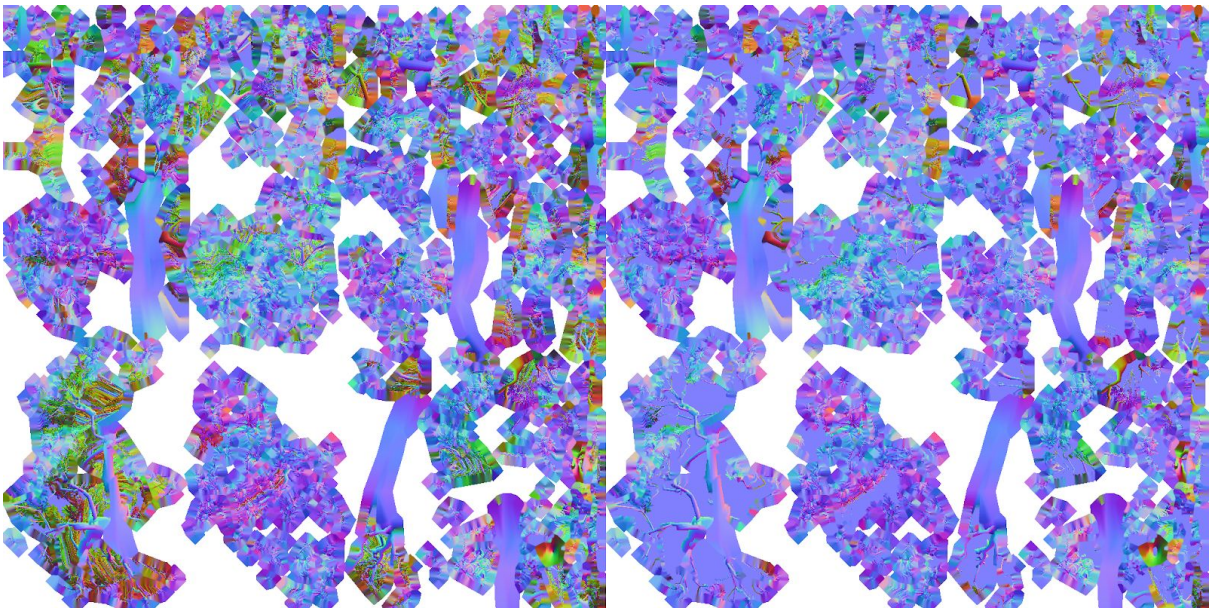


Figura 5.17. A l'esquerra: normal map sense especificar distància. A la dreta: normal map amb distància de 0'5.

En models amb molta geometria interna com per exemple els arbres, ens pot interessar fer servir valors de distància petits per evitar que al fer el *bake* valors de normals o textures d'elements que són molt lluny d'una cara o fins i tot a l'altre banda del model afectin als valors que s'hi acaben assignant.

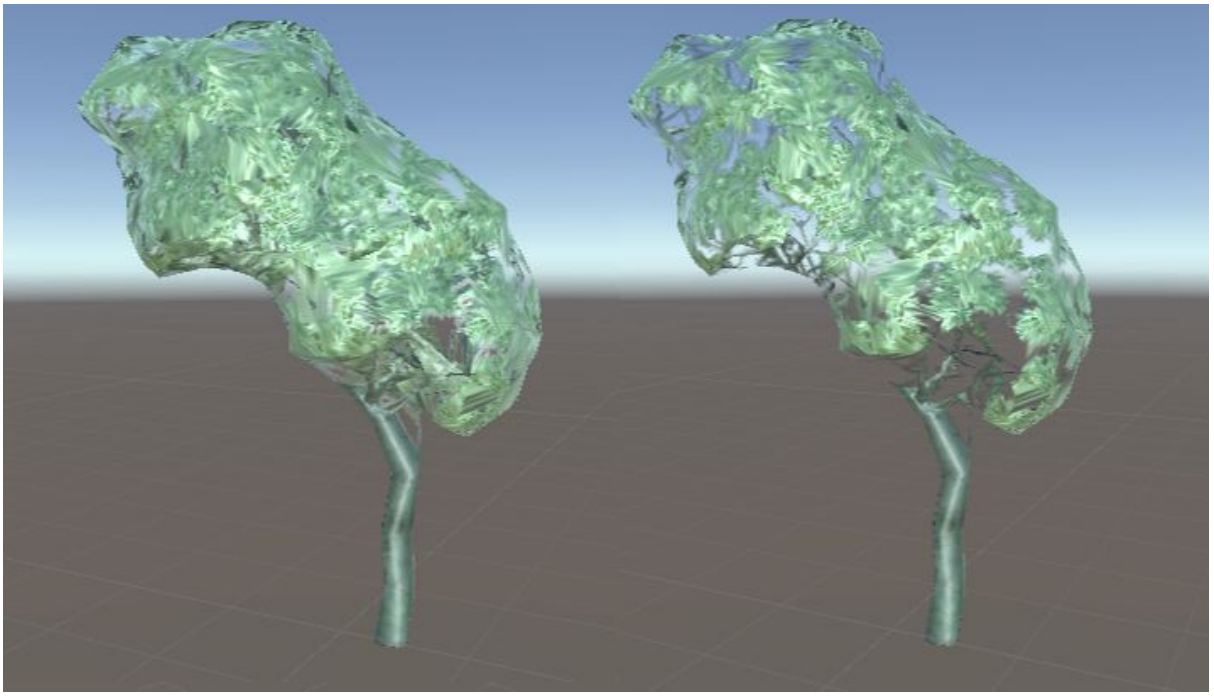


Figura 5.18. A l'esquerra: model simplificat amb baking de textures i normals sense especificar distància. A la dreta: model simplificat amb baking de textures i normals amb distància de 0'5.

Per a fer el *bake* de les textures del model original al model simplificat seguim els mateixos passos que per a les normals. Si volem capturar la topologia del model original a través d'aquest pas haurem d'afegir uns passos addicionals.

Per aconseguir capturar la topologia del model original a través del *bake* de textures hem de afegir el canal *alpha* a la textura associada al model simplificat amb valor a zero. També hem de activar la transparència al material associat al model simplificat i deixar a zero els valors alpha i especular.

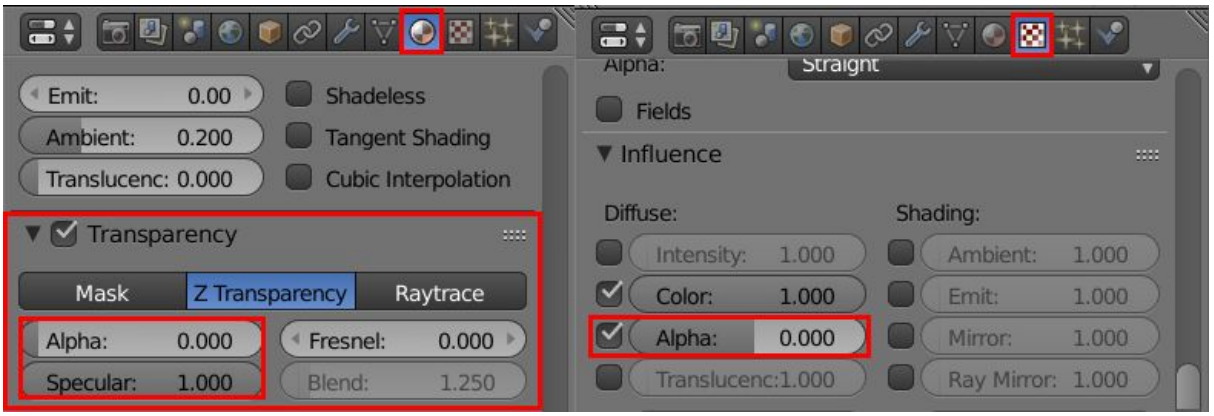


Figura 5.19. Preparar la malla simplificada per obtenir transparències a les textures resultants del bake.

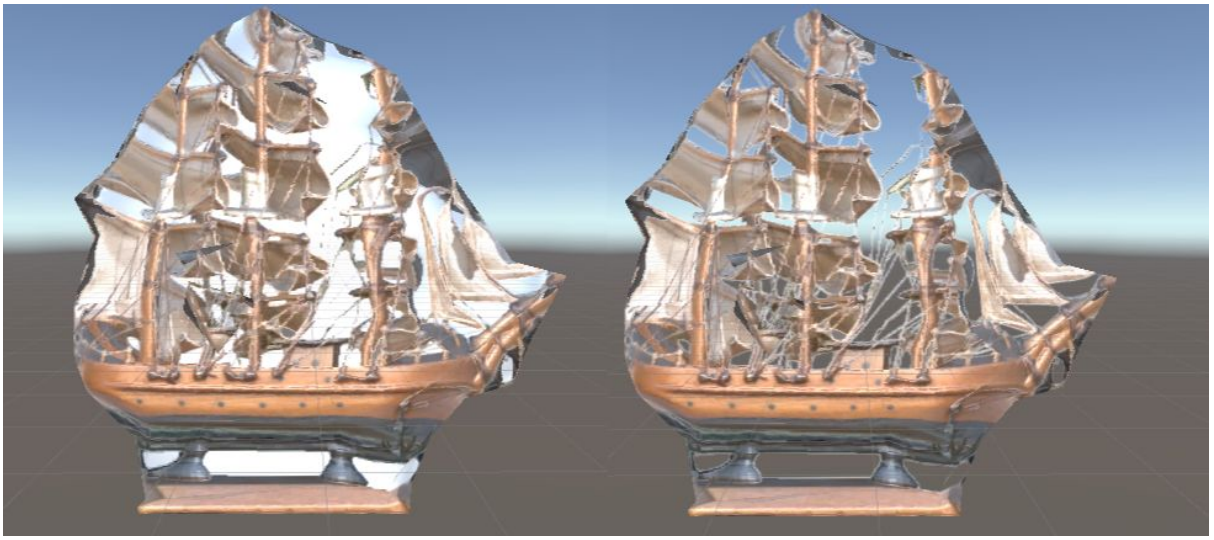


Figura 5.20. A l'esquerra: model amb textura sense transparències. A la dreta: model amb textura amb transparències.

6. Resultats

En aquest apartat mostrarem els resultats obtinguts d'aplicar l'algorisme a diferents models on els algorismes tradicionals obtenen pitjors resultats. Ens centrarem en observar la sortida obtinguda en cada pas del *pipeline* de l'algorisme i identificar en quins casos funciona i en quins no i perquè.

Apart, hem realitzat una demo amb un dels casos d'èxit, concretament amb el model "High Poly Tree". La demo consisteix en un petit programa que renderitza una escena amb 217 models de l'arbre i una càmera que es va movent per l'escena realitzant un recorregut concret.

Els models que s'utilitzen són el model original del "High Poly Tree" i la seva simplificació de 1000 cares. Més informació sobre els models a l'Annex I.

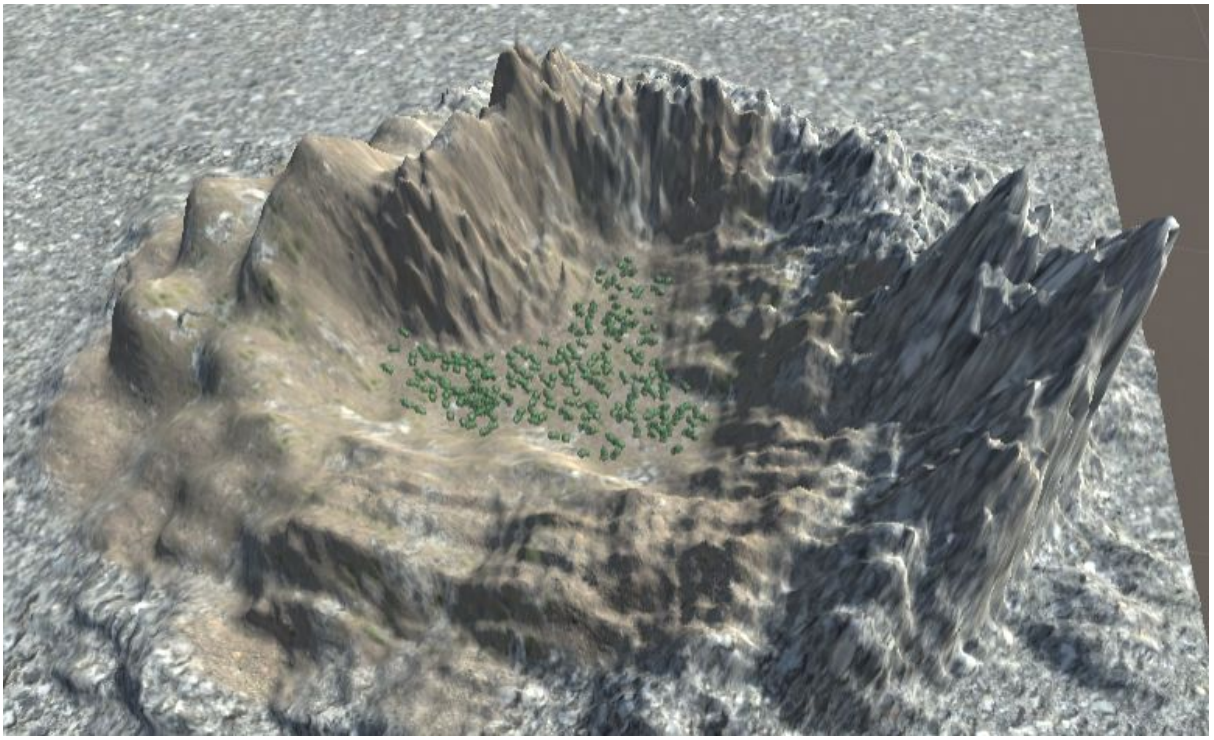


Figura 6.1. Pla general de l'escena de la demo.

Durant aquest recorregut s'allunya i apropa del conjunt de models, enfocant-los com a pla general allunat i passant a través d'ells. Així podem comprovar com respon la demo en les diferents situacions possibles de renderitzat d'una escena.

L'objectiu d'aquesta demo es comprovar el seu rendiment utilitzant les simplificacions del model a través de la tècnica *LOD* respecte el rendiment de la mateixa demo utilitzant models sense simplificar. Per a fer-ho la demo calcula el *framerate* a cada *frame* i deixa els resultats en un fitxer .csv.

D’aquests fitxers n’hem obtingut les següents gràfiques que ens permeten analitzar els resultats:

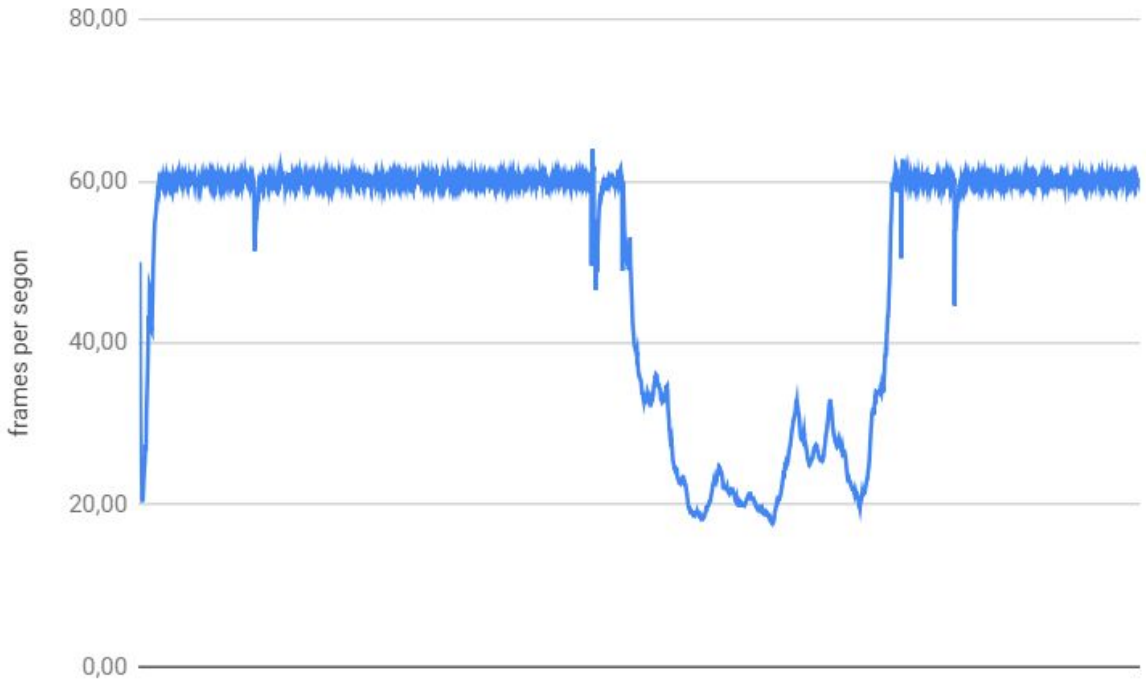


Figura 6.3. framerate de la demo amb LOD.

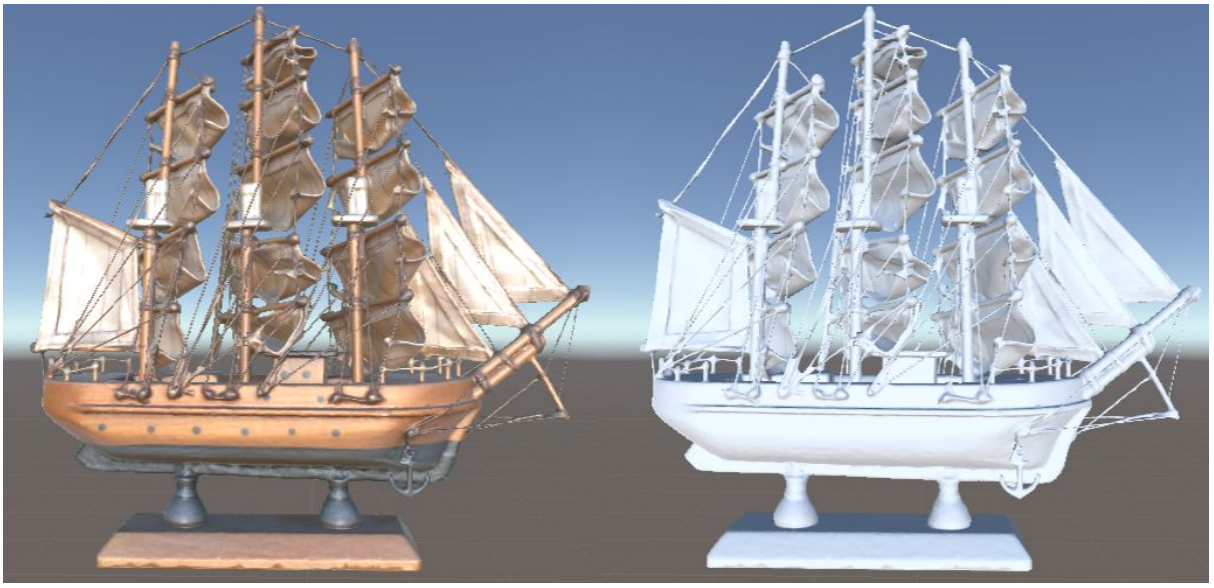
En la gràfica anterior podem observar com hi ha dues franges a l’inici i el final de la demo on el framerate es manté a 60fps, coincidint en els moments on la càmera està realitzant plans generals i allunyats de l’escena i es renderitxen els models simplificats. Quan la càmera s’apropa i es comencen a renderitzar alguns models sense simplificar el framerate va baixant progressivament fins arribar a tocar fons en certs moments a 20fps.

En canvi la demo sense les tècniques de *LOD* en cap moment el seu framerate passa de 10fps, ja que ha de renderitzar tota l'estona els models originals cosa que és altament ineficient tal com mostren els resultats.

Abans de mostrar les imatges dels resultats sobre els models, volem recordar que l'objectiu d'aquest projecte és obtenir simplificacions per als models que es troben més allunyats de l'escena i que l'ull humà no pot veure amb tanta claredat com els que hi ha en primer pla. Així doncs, tot i que en les imatges que us presentem a continuació les simplificacions es troben en primer pla, hem de tenir en compte que no és pel que estan pensades ni l'objectiu que persegueix aquest projecte.

6.1. Sailing ship

Aquest model és especialment interessant per aquest projecte ja que es tracta d'un model construït a partir d'un escanejat 3D de l'objecte real. Té gran quantitat de detalls a la part superior, màstils, velam i cordes, que generen una topologia molt complexa.



6.4. Model original.

El primer que farem serà observar els resultats obtinguts d'aplicar els algorismes *quadrics* i *vertex clustering* al model original.

Figura 6.5. D'esquerra a dreta: simplificacions amb quadrics de 1000, 500 i 250 cares respectivament.

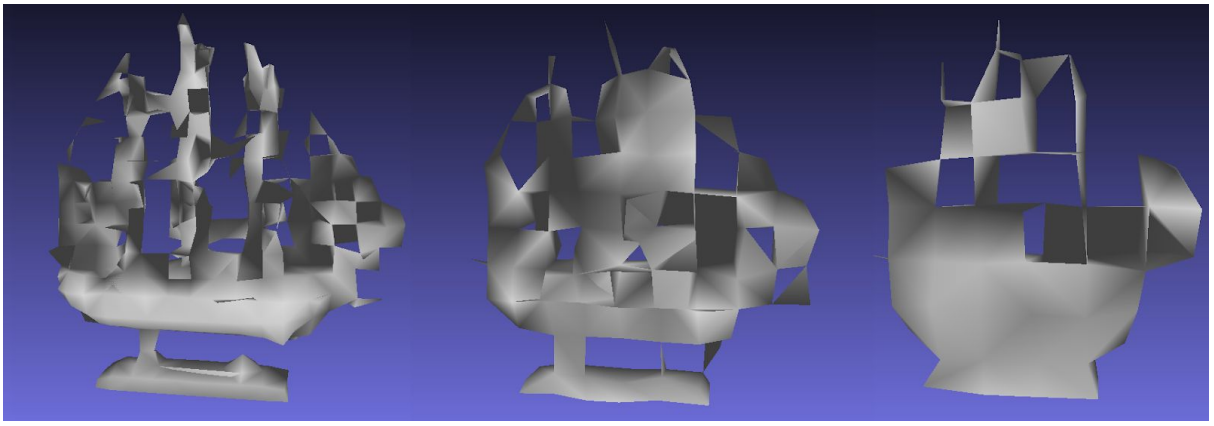


Figura 6.6. D'esquerra a dreta: simplificacions amb vertex clustering de 2060, 730 i 313 cares.

Podem comprovar a partir de les figures 6.5 i 6.6 que els resultats obtinguts són molt poc similars al model original i han perdut geometria o generat nous forats a la malla simplificada.

Ara mostrarem els resultats obtinguts a cad pas del nostre algorisme amb aquest model, així com el resultat final.

Figura 6.7. A l'esquerra model amb les cares normalitzdes. A la dreta: alpha shape del model.

En la figura anterior podem veure com l'*alpha shape* aconsegueix ajustara-se força al model normalitzat, no obstant ha tancat les zones del peu del model. Cosa que en passos posteriors provocarà problemes.

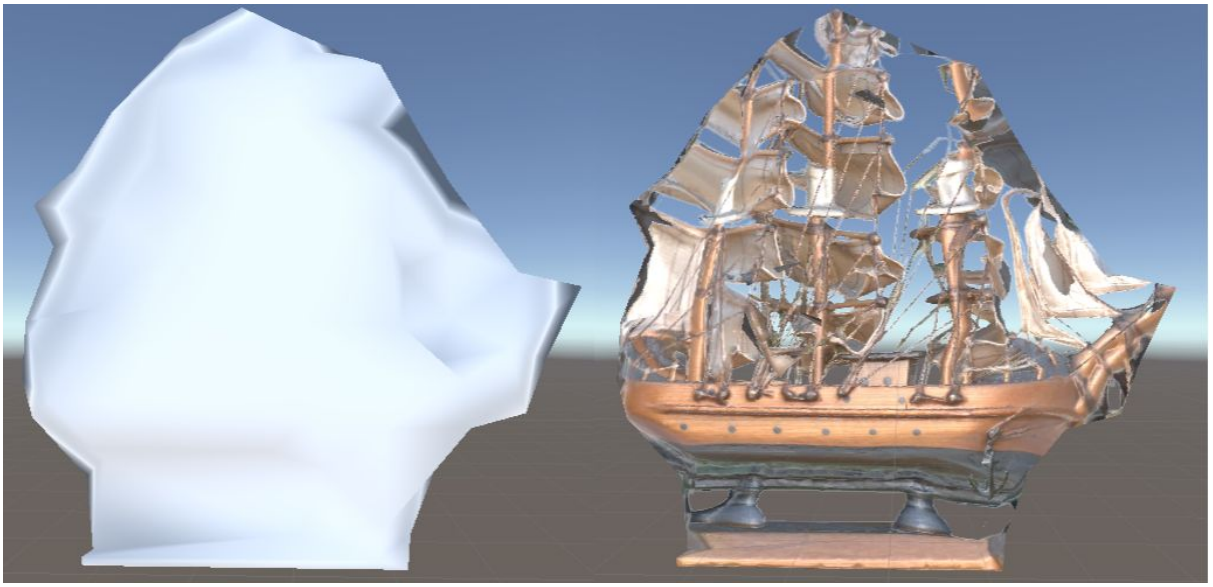


Figura 6.8. A l'esquerra model simplificat de 250 cares. A la dreta: el mateix model amb els mapes de textures i normals.

Figura 6.9. A l'esquerra: normal map del model simplificat. A la dreta: Textures del model simplificat

En les figures anteriors podem veure el model simplificat amb i sense el mapeig de normals i textures. Podem veure clarament com sense el postprocessat la simplificació careix de semblança amb el model original.

També apreciem els problemes esmentats anteriorment al peu del model, on les textures s'han deformat a causa de l'aparició de nova geometria al model. I altres deformacions als límits del model. En aquest cas no hem pogut jugar amb el paràmetre de distància del *baking* ja que sinó es perden els detalls centrals com els màstils.

Malgrat això, la simplificació obtinguda no deixa de representar cap part del model original i hi manté una certa semblança.

6.2. Cupboard

Aquest model també és interessant per aquest projecte. Es tracta d'un model d'un armari amb tot d'elements als seus prestatges. Cada un d'aquests elements no està connectat a la resta de la malla i n'hi ha que són més complexos que d'altres. Com abans, primer mostrarem el model original i les simplificacions obtingudes amb els algorismes actuals.

Figura 6.10. Model original

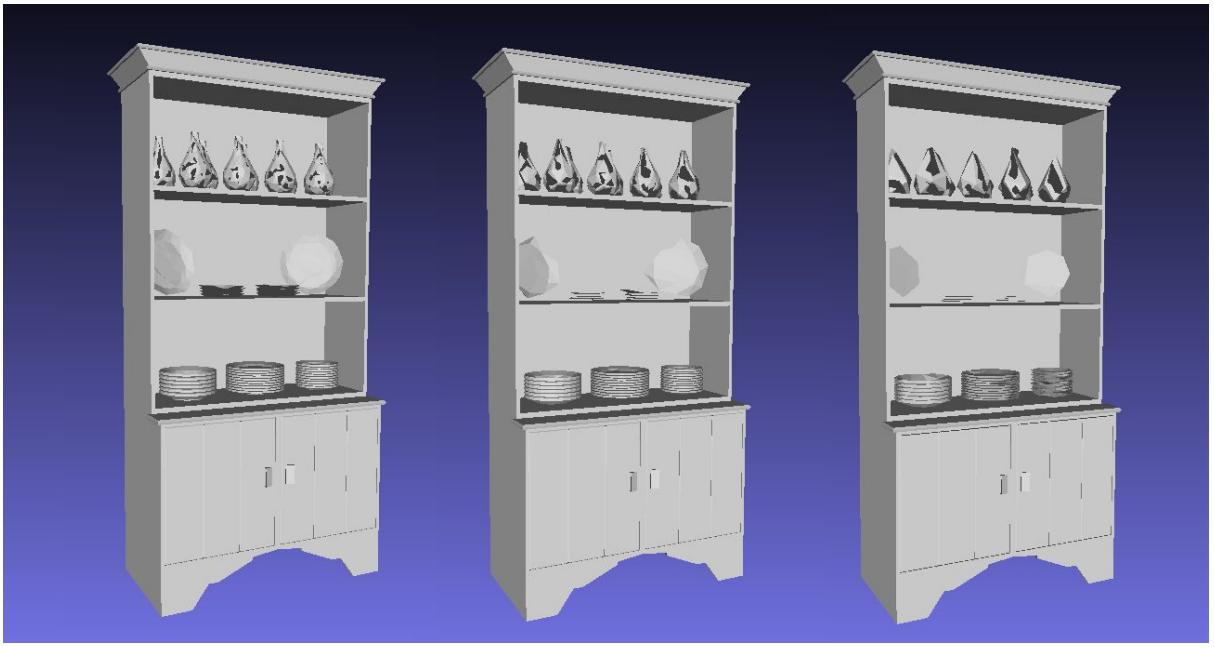


Figura 6.11. D'esquerra a dreta. Simplificacions amb quadrics de 10000, 5000 i 2500 cares.

Figura 6.12. D'esquerra a dreta. Simplificacions amb vertex clustering de 11248, 5230 i 2487 cares

Tot i que les simplificacions obtingudes amb els algorismes actuals no són dolentes en general, podem veure com els elements dels prestatges perden geometria a mida que la simplificació és major. Per tant, mirem ara com es comporta el nostre algorisme.

Figura 6.13. A l'esquerra model normalitzat. A la dreta: Alpha shape del model

Figura 6.14. A l'esquerra model simplificat amb 2500 cares. A la dreta el mateix model amb normals i textures.

Al model simplificat resultant podem observar que la part del model on hi ha els gerros ha quedat força mal simplificada en comparació amb la resta del model simplificat. Això es deu a que en el pas de l'*alpha shape* el valor d'*alpha* hauria hagut de prendre un valor més gran per a poder crear bé l'*alpha shape* d'aquella zona. Però quan pren valors més grans la superfície ja tanca el prestage del gerros obtenint una simplificació encara pitjor.

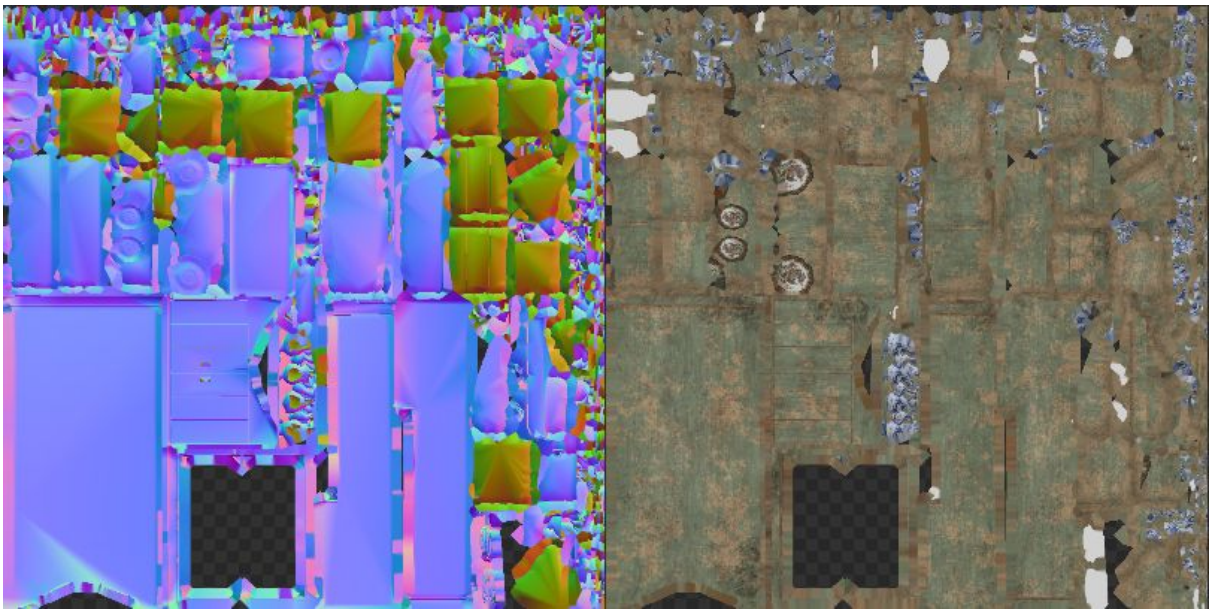


Figura 6.15. A l'esquerra: normal map del model simplificar. A la dreta: textures del model simplificat.

6.3. High Poly Tree

El model d'arbre va ser el primer que ens va venir en ment quan vem iniciar aquest projecte. Aquest model en concret està tot format per geometria i les fulles no són plans texturats. Així doncs, es tracta d'un model amb molta geometria i una topologia molt complexa. també és un model amb molta densitat. Per tant és un tipus de model on el nostre algorisme hauria de funcionar millor.

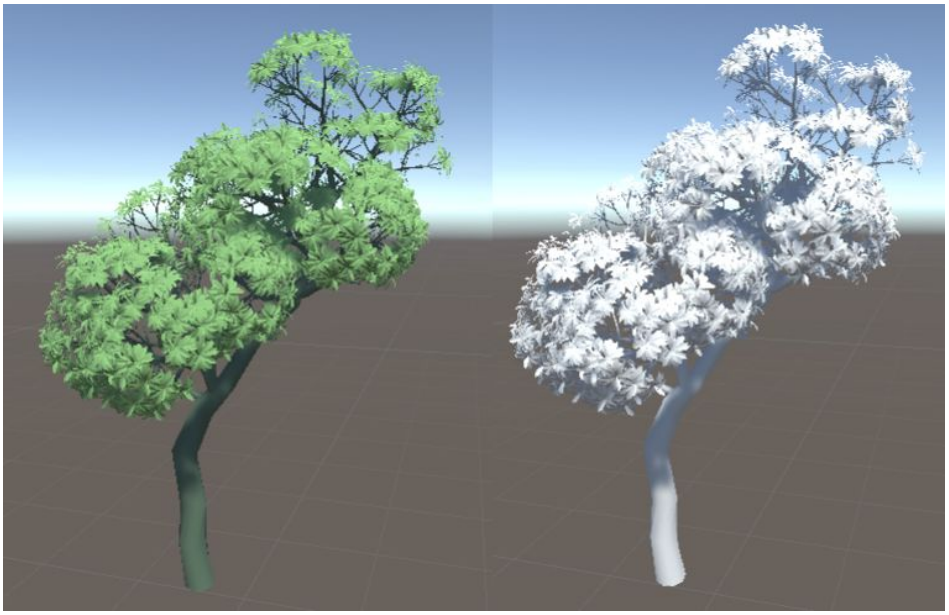


Figura 6.16. Model original.

Si observem els resultats dels algorismes de simplificació actuals sobre aquest model podem veure que els resultats són absolutament inacceptables, amb pèrdua de moltíssima geometria o grans deformacions respecte al model original.

Figura 6.17. Simplificacions amb quadrics de 1000, 500 i 250 cares.

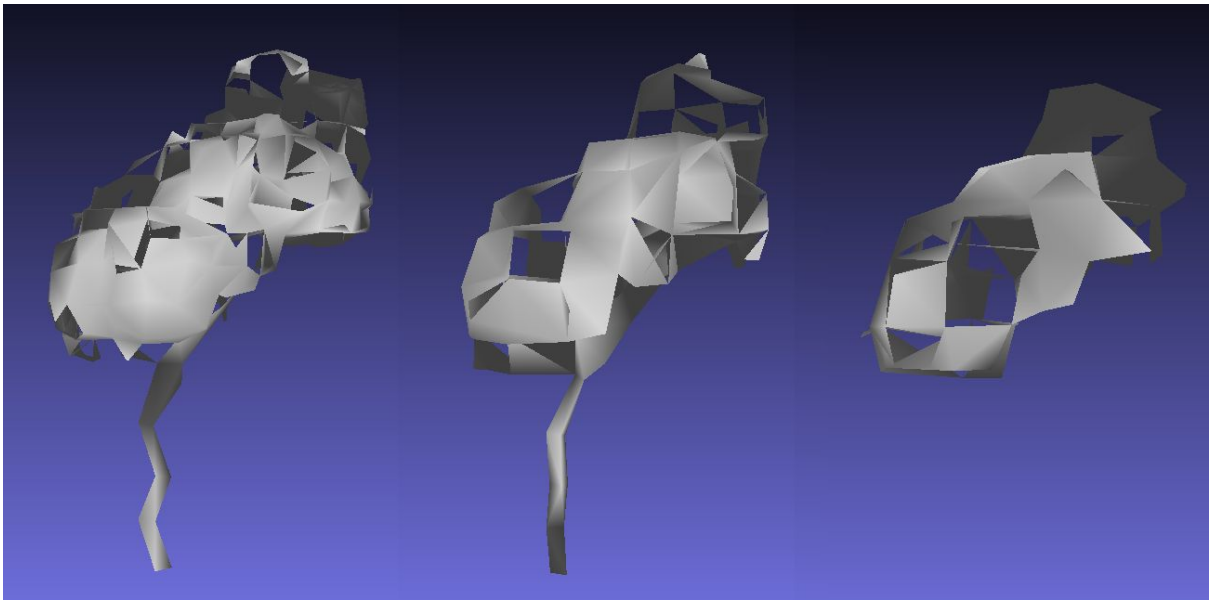


Figura 6.18. Simplificacions amb clustering de 1507, 479 i 331 cares.

En aquest model hem optat per no aplicar el preprocessat de normalització de del model d'entrada ja que provoca pèrdua de geometria similar al les simplificacions anteriors i el valor d'*alpha* que fem servir és prou gran com per a que no pugui crear forats a la malla. Així doncs, els resultats obtinguts en aquest model són el següents.

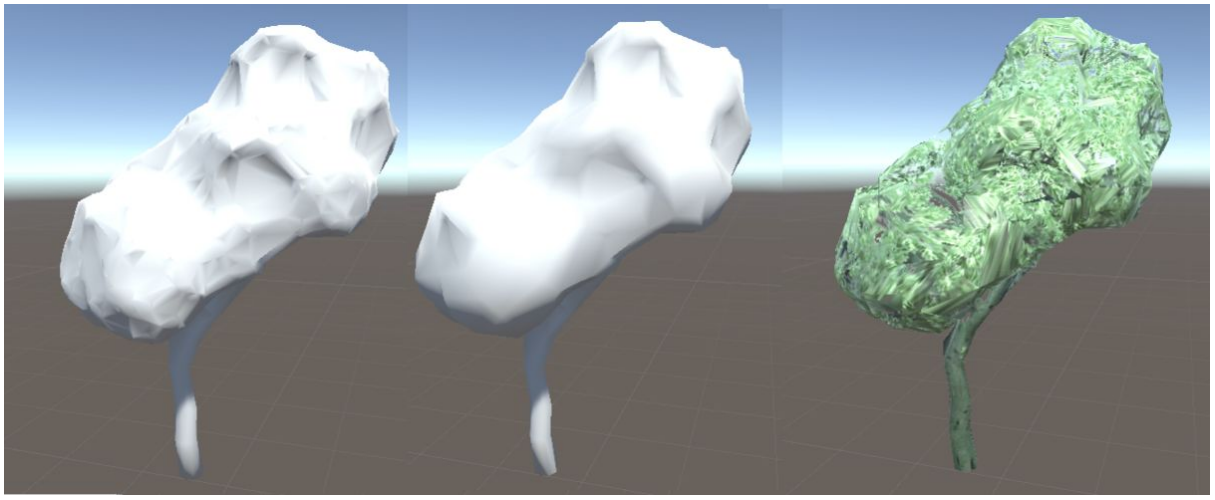


Figura 6.19. D'esquerra a dreta. Alpha shape del model, model simplificat a 1000 cares i la simplificació amb normals i textures.

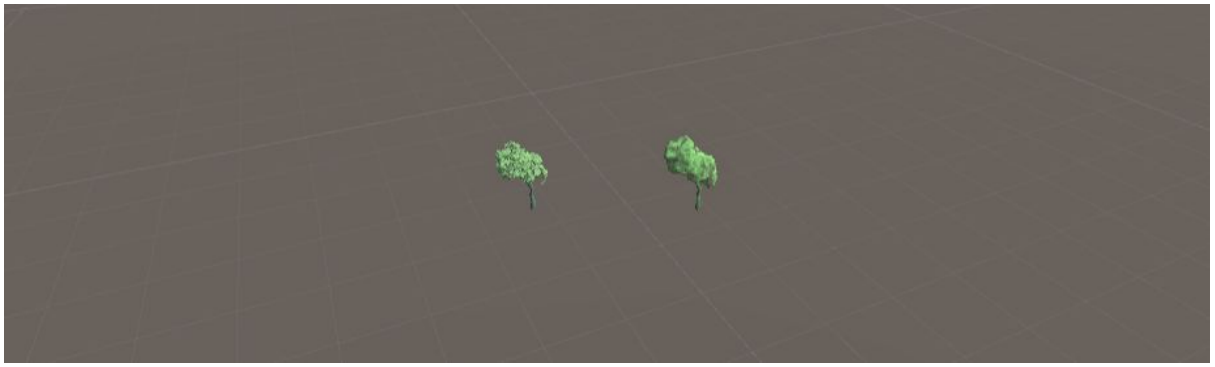


Figura 6.20. Model original, a l'esquerra, i model simplificat, a la dreta, al fons de l'escena.

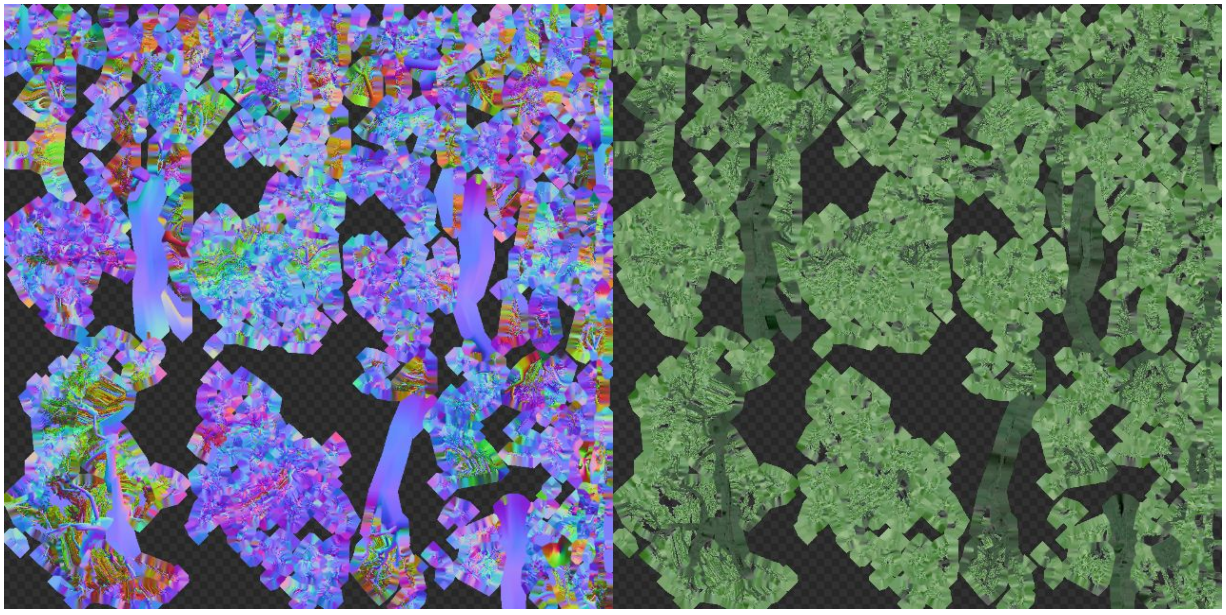


Figura 6.21. A l'esquerra: normal map del model simplificat. A la dreta: textures del model simplificat.

A la nostra simplificació podem veure que no es perd geometria però, com que el model original té tanta geometria és difícil recuperar-ne la topologia.

Tot i això, podem veure a la figura 6.20 que és una simplificació molt millor que les que podem obtenir amb els algorismes actuals.

6.4. Charme

Aquest model és el més simple dels que hem tractar amb l'algorisme que proposem. Tot i així, la seva estructura general en forma esfèrica i el fet de que el model, tot i tenir molta topologia, forma una superfície envolupant clara, l'han fet ideal per a provar l'algorisme.

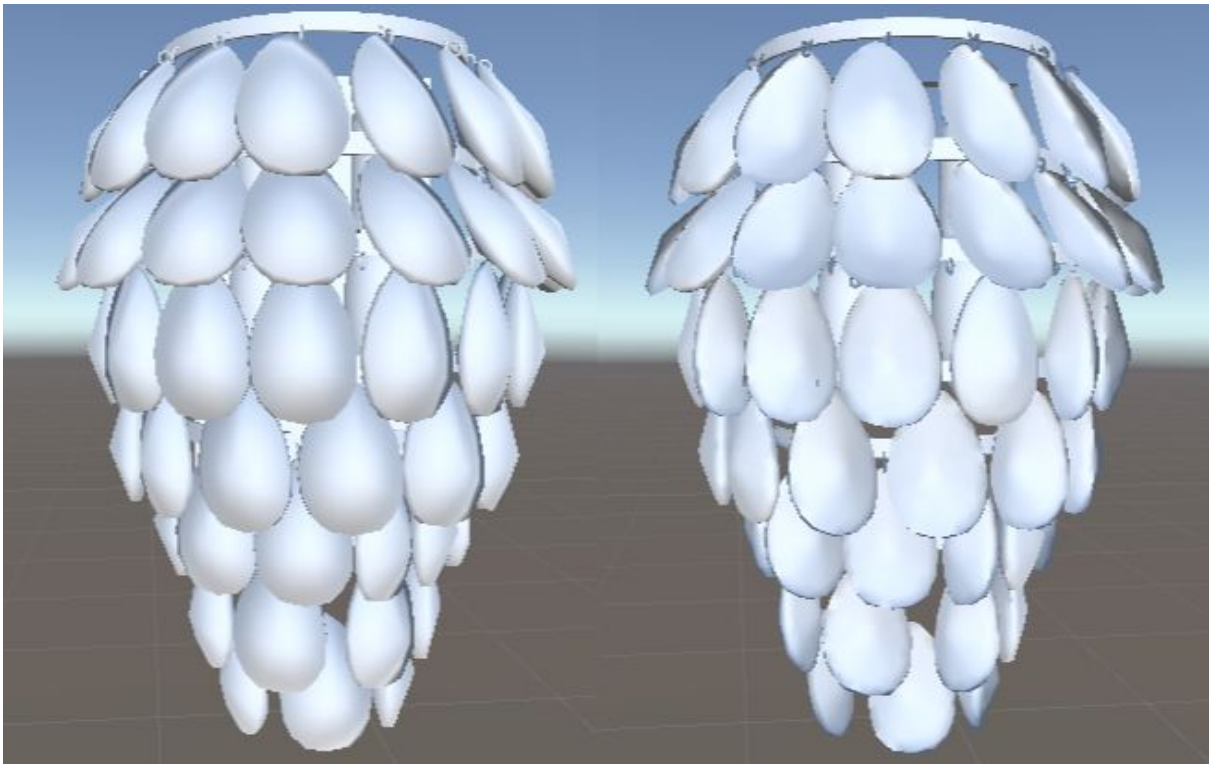


Figura 6.22. A l'esquerra, Model original. A la dreta. Model normalitzat.

En la figura anterior podem veure com el model normalitzar gairebé no difereix del model original. No obstant, aquest preprocés ha creat moltes cares respecte el model original.

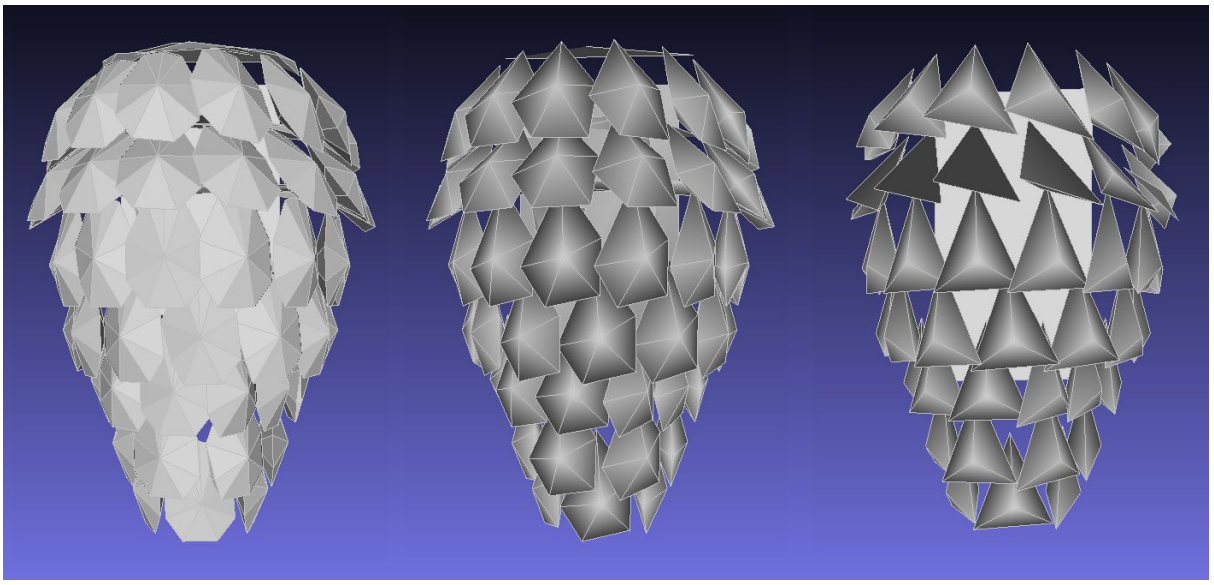


Figura 6.23. Models simplificats amb quadrics amb 1000, 500 i 250 cares.

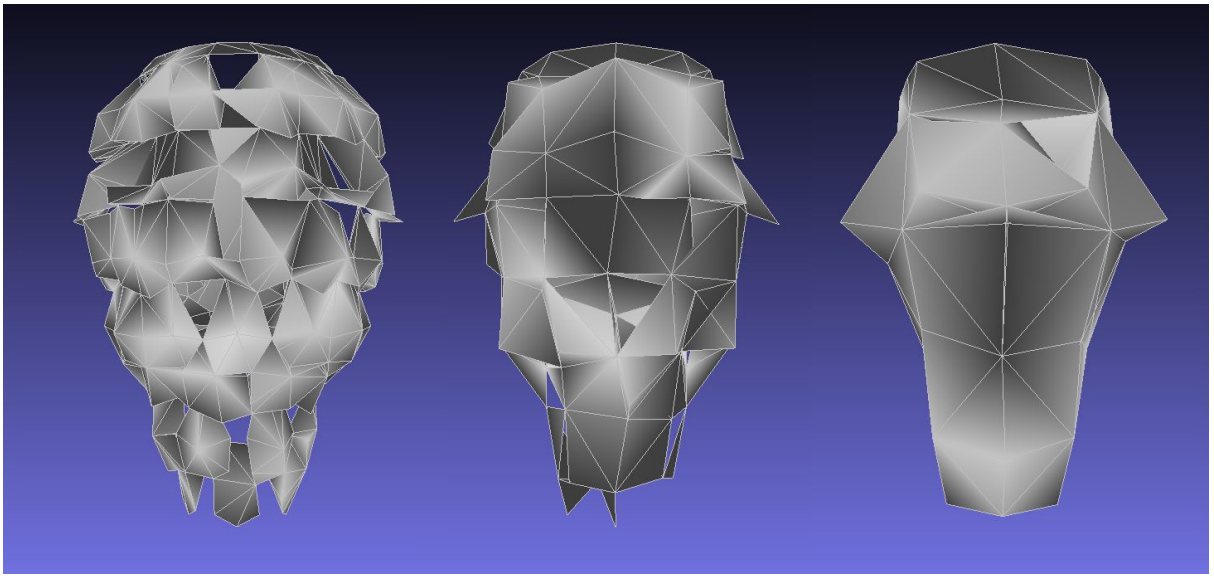


Figura 6.24. Models simplificats amb vertex clustering amb 1075, 362 i 140 cares.

De nou, els algorismes actuals aconsegueixen simplificacions poc semblants al model original, sobretot quan volem simplificacions més grans. En canvi a la següent figura podem observar la simplificació que hem obtingut amb el nostre algorisme.

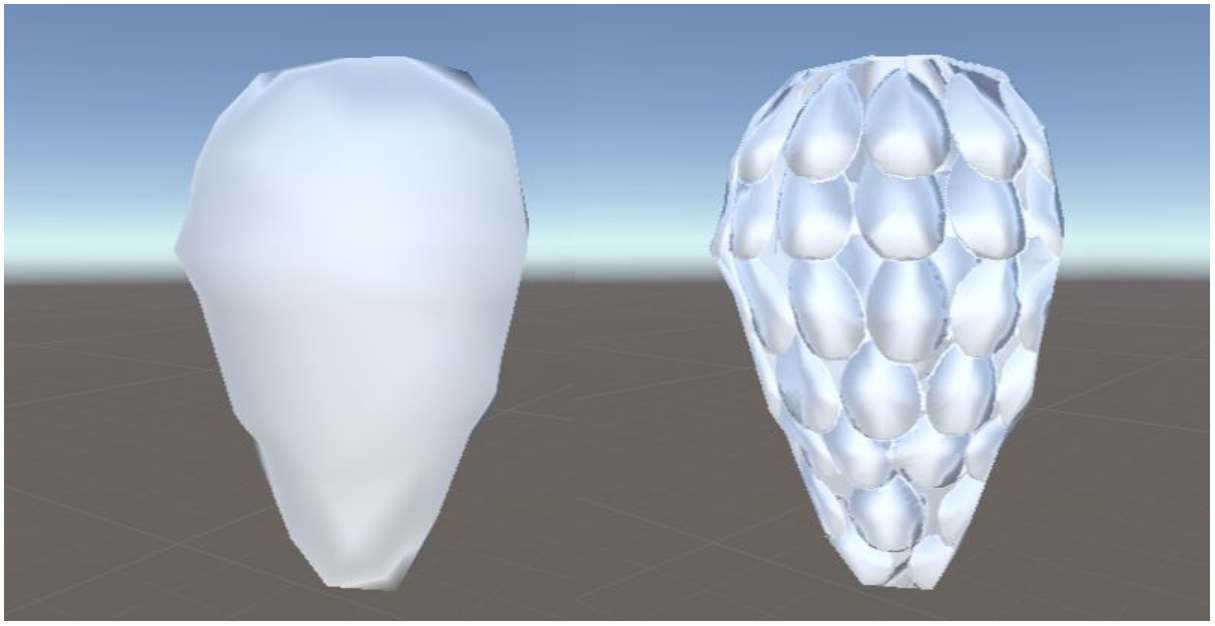


Figura 6.25. D'esquerra a dreta. Simplificació de 250 cares i la mateixa simplificació amb normal map.

Si bé és cert que la malla simplificada ha perdut detalls que no hem pogut recuperar amb el *normal map*, la simplificació continua sent millor que les obtingudes amb els altres algorismes.

6.5. Casos incorrectes

Apart de mostra-vos els casos d'èxit de l'algorisme també us volem mostrar el casos de fracàs, on l'algorisme no ha aconseguit obtenir una simplificació prou similar al model original com per a que puguem considerar-la correcte.

Volem remarcar però que, tot i no aconseguir models similars als originals, les simplificacions obtingudes mantenen la forma del model original. En canvi, els algorismes actuals perden geometria quan els simplifiquen.

6.5.1. Stalks of corn

Aquest model podem dir que és similar al cas de l'arbre. Es tracta d'un model amb molta geometria ramificada. Tot i així, la poca densitat d'algunes zones del model provoquen que l'*alpha shape* crei una superfície molt allunyada del model original provocant deformacions a la malla que acaben desembocant a una simplificació molt poc similar al model original.

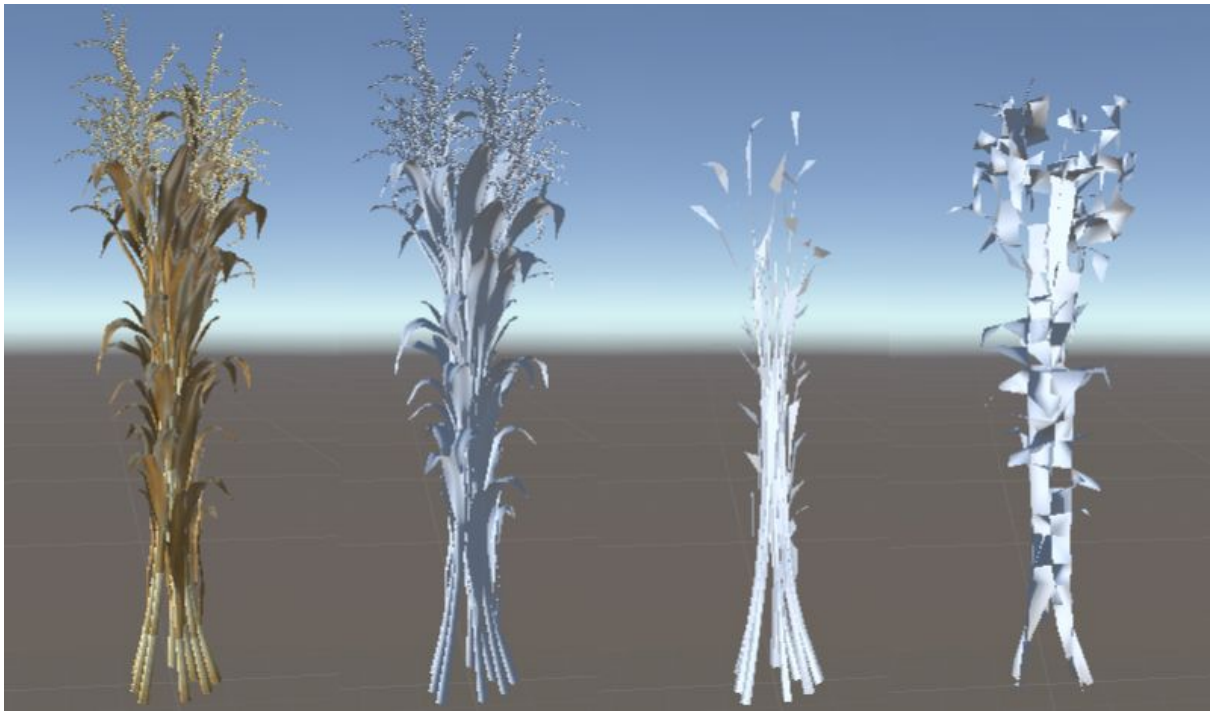


Figura 6.26. D'esquerra a dreta. Model original amb i sense textures, simplificat amb quadrics a 1000 cares i model simplificat amb vertex clustering a 1923 cares.

A la figura anterior podem veure que els algorismes actuals tampoc són capaços de generar una bona simplificació del model. Mirem ara però la simplificació que obtenim utilitzant el nostre algorisme.

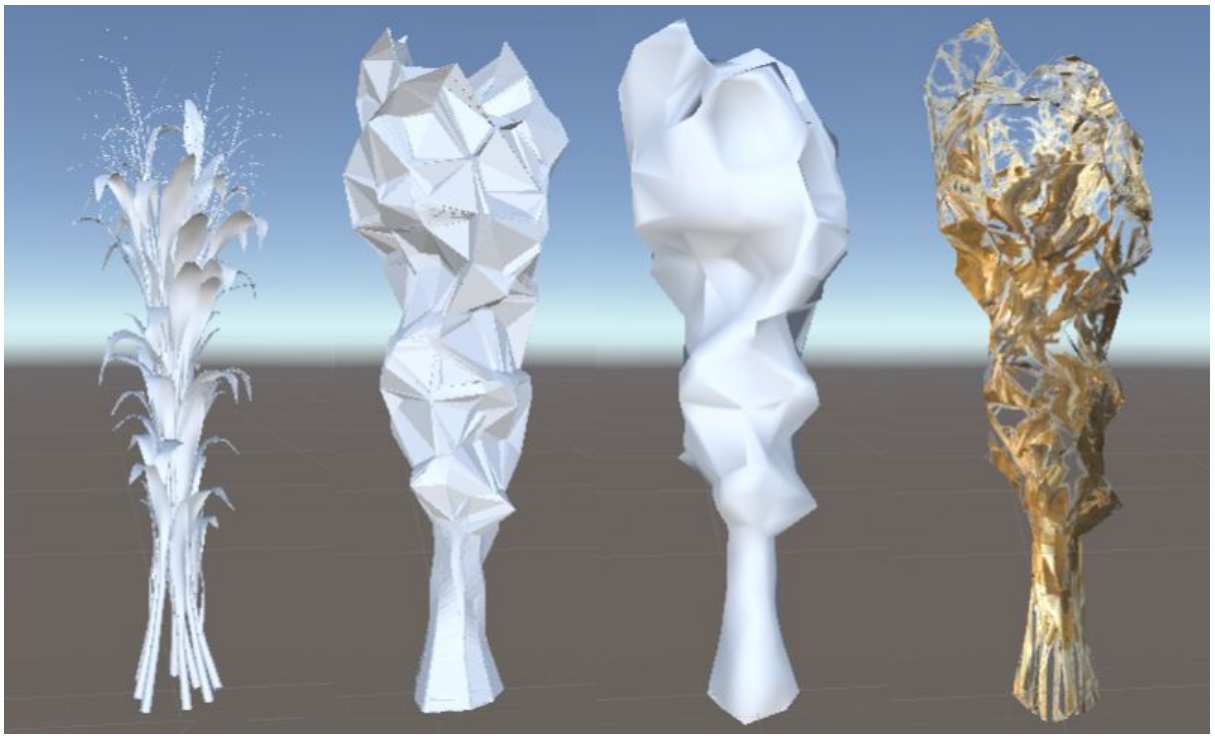


Figura 6.27. D'esquerra a dreta. Model normalitzat, model d'alpha shape, model simplificat a 1000 cares i el mateix model simplificat amb textures i normals.

Ja d'entrada podem observar que la normalització ha causat pèrdua de geometria a la part superior del model on hi havia representat gra. Després, l'*alpha shape* té molts problemes per a generar una superfície pròxima al model normalitzat i alhora tancar prouta topologia per a que després la simplificació obtingui bons resultats.

Això provoca que l'*alpha shape* generi molta geometria allunyada del model original que després quan es fa el mapeig de les textures i normals del model original al simplificat provoca greus deformacions. Obtenint així un resultat final que té molt poca semblança amb el model original.

6.5.2. House Plant

Aquest model és similar a l'anterior, els algorismes actuals no són capaços de donar bones simplificacions i d'entrada semblava un bon model per aplicar-hi l'algorisme. Després de fer-hi probes, ens hem acabat trobant amb els mateixos problemes que a "*Stalks of corn*".

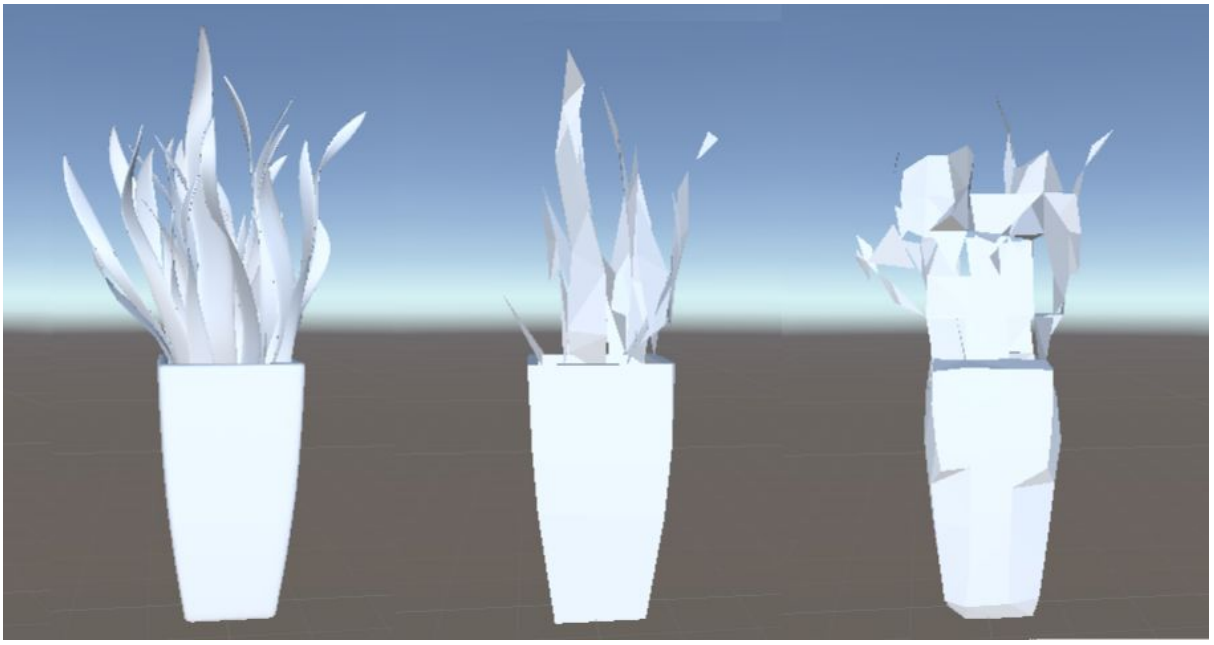


Figura 6.28. D'esquerra a dreta. Model original, simplificació amb quadrics a 250 cares i amb clustering a 648.

Concretament el problema és una geometria que a la part superior del model deixa de ser prou densa com per a poder fer un bon pas d'*alpha shape* que després acaba limitant les resta de l'algorisme.

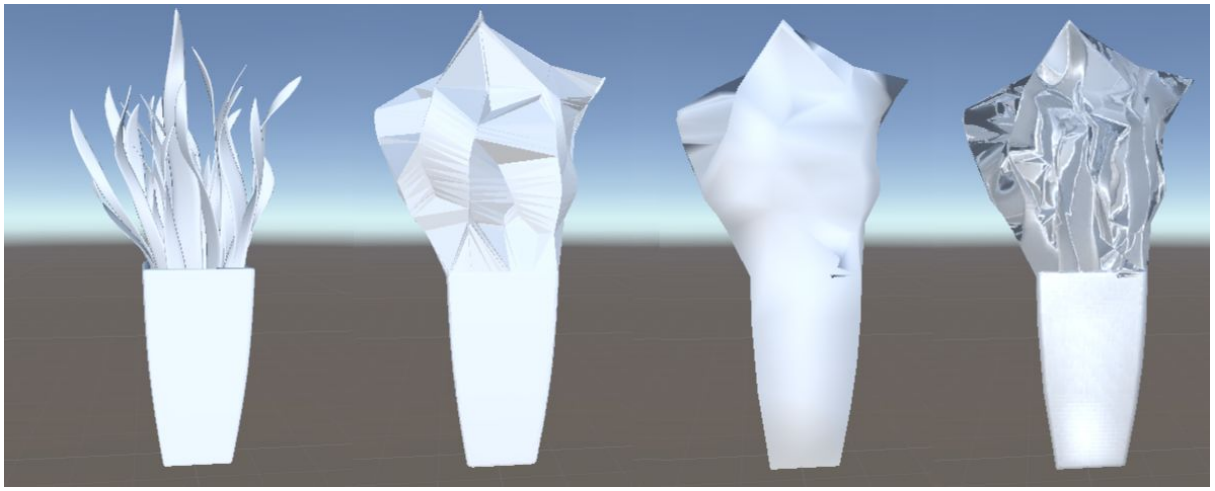


Figura 6.29. D'esquerra a dreta. Model normalitzat, model d'alpha shape, model simplificat a 250 cares i model simplificat amb normal map.

Igual que en el cas anterior l'alpha shape genera la superfície molt allunyada del model original i llavors en el post processat quan mapegem les normals provoca deformacions al mapeig.

7. Gestió del projecte

7.1. Planificació

Aquest projecte es tracta d'un estudi de viabilitat d'un algorisme, a causa d'axió el projecte ha canviat considerablement del plantejament inicial al que ha estat realment. En un inici es va plantejar la possibilitat d'arribar a implementar part del *pipeline* o fins i tot tot el *pipeline*, però després d'obtenir els primers resultats es va descartar aquesta idea i ens vem centrar en afegir el postprocessat a la tècnica per aconseguir més semblança visual entre el model original i el simplificat.

Això es deu a que cada pas que realitzavem trobàvem imprevistos que van allargar les tasques d'avaluació de l'algorisme proposat. Així doncs, la planificació inicial s'ha vist totalment alterada perdent totes les tasques d'implementació i afegint tasques de postprocessat que vem marcar com opcional a l'inici del projecte. També hem documentat i realitzat tests a cada pas del *pipeline*.

	Nom de la tasca	Inici	Final	Durada	Predecessors
1	**Investigació**	**12/09/18**	**18/09/18**	**5d**	
2	**Planificació del progecte (GEP)**	**19/09/18**	**15/10/18**	**19d**	**1**
3	Definició d'abast i contextualització	19/09/18	25/09/18	5d	
4	Planificació temporal	26/09/18	01/10/18	4d	3
5	Gestió econòmica i sotenibilitat	02/10/18	08/10/18	5d	4
6	Revisió i correcció	09/10/18	15/10/18	5d	5
7	**Avaluació de l'algorisme proposat**	**16/10/18**	**12/11/18**	**20d**	**2**
8	Cerca de models de proves	16/10/18	18/10/18	3d	
9	Estudi dels programes de prova	19/10/18	24/10/18	4d	8
10	Programació del pas alpha shape R	25/10/18	05/11/18	8d	9
11	Lectura de models en R	25/10/18	30/10/18	4d	
12	Escriptura de models en R	31/10/18	05/11/18	4d	11
13	Proves de l'algorisme sobre els models	06/11/18	09/11/18	4d	10
14	Documentació dels resultats	12/11/18	12/11/18	1d	13
15	**Estudi dels frameworks a utilitzar**	**13/11/18**	**23/11/18**	**9d**	**7**
16	Estudi decimation framework (OpenMesh)	13/11/18	16/11/18	4d	
17	Estudi 3D Alpha shapes (CGAL)	19/11/18	23/11/18	5d	16
18	**Implementació**	**26/11/18**	**09/01/19**	**33d**	**15**
19	Lectura i escriptura de models de malles	26/11/18	29/11/18	4d	
20	Alpha Shape del modle normalitzat	30/11/18	05/12/18	4d	19
21	Simplificació amb Quadrics	06/12/18	11/12/18	4d	19; 20
22	Normalització de distàncies entre vèrtex	12/12/18	17/12/18	4d	21
23	Unió del pipeline	18/12/18	26/12/18	7d	19; 20; 21;
24	Afegir propietats de la malla original a la simplificada (estudi)	27/12/18	31/12/18	3d	23
25	Afegir propietats de la malla original a la simplificada (implementació)	01/01/19	09/01/19	7d	24
26	Testos sobre els models de proves	10/01/19	15/01/19	4d	18
27	Documentació dels resultats	16/01/19	17/01/19	2d	26

Figura 7.1. Planificació inicial del projecte.

També s'han incorporat i ampliat les tasques de tests sobre els models de prova, comprovant els resultats obtinguts a cada pas de l'algorisme i en el postprocessat que

estava previst com opcional a l'inicia del projecte. I la realització d'una demo que demostra l'èxit de l'algorisme.

Podem concloure doncs que la planificació inicial no s'ha pogut seguir a causa del tipus de projecte que hem dut a terme però hem re-planificat el projecte per a poder provar la viabilitat de l'algorisme.

7.1. Dimensió Econòmica: Anàlisi

El càlcul dels costos del projecte realitzats a l'inici d'aquest no han resultat ser correcte ja que la planificació del projecte ha canviat totalment del que es va plantejar a l'inici. Tot i que el nombre d'hores empleades al projecte han estat les mateixes no s'ha assolit la implementació del *pipeline* de l'algorisme, cosa que ens produeix una desviació dels costos importants.

Pel que fa a vida útil del producte creiem que no té sentit parlar de vida útil ja que es tracta d'un algorisme. L'algorisme és força més lent que els algorismes actuals a causa dels diversos passos que ha de realitzar arribant a tardar uns 27 segons a calcular la simplificació del model "High_Poly_Tree", snese tenir en compte el postprocessat. Més resultats a l'Annex I.

7.2. Dimensió Ambiental: Anàlisi

El cost ambiental del projecte l'hem estimat a través del consum de l'ordinador de sobretaula i altres elements d'oficina necessaris, i la durada prevista del projecte. Tot i que la planificació del projecte ha canviat totalment durant el transcurs d'aquest el nombre total d'hores destinades no s'ha vist afectat.

Així doncs, el cost ambiental previst és el següent:

Concepte	**Consum**	**Durada**	**Cost**
Ordinador sobretaula	450 W	456 hores	205 KWh
Monitor	75 W	456 hores	34 KWh
Mòdem	30 W	456 hores	14 KWh
Total			**253 KWh**

Figura 7.2.. Cost ambiental del projecte

Aquest cost creiem que ja no es pot reduir tenint en compte que només fem servir els recursos que ens són imprescindibles per a desenvolupar el projecte. L'únic que ens permetria reduir el cost seria utilitzar un ordinador que tingui menys consum, però pel tipus de projecte que duem a terme creiem necessari utilitzar un ordinador modern amb bona potència de càlcul.

Podem afirmar que el nostre algorisme ha generat models molt més simples que les originals. Per exemple en el cas del model "High_Poly_Tree" la simplificació utlitzada en les demos té fins a 122 vegades menys vèrtex que el model original. Més resultats a l'Annex I.

Pel que fa a altres tipus de contaminació aquest projecte és totalment net, ja que no utilitza matèries primeres (apart de l'electricitat) i no consisteix en fabricar cap producte material que després es distribueixi físicament.

7.3. Dimensió Social: Anàlisi

Socialment parlant, el projecte no hem previst que tingui cap impacte significatiu, millora de la qualitat de vida o controversia social entre la societat i el gran públic. Estrictament parlant la societat no té necessitat de la realització d'aquest projecte, ja que es tracta de la millora d'una tècnica que la majoria de la gent passa per alt.

Els col·lectius que més poden utilitzar o notar els efectes del projecte són altres persones que facin recerca en aquest camp, desenvolupadors d'aplicacions gràfiques i en menor mesura els consumidors d'aquestes aplicacions gràfiques. Però és en els desenvolupadors on volem que tingui més impacte el producte resultant d'aquest projecte, permetent-los tenir una eina que els permeti solucionar problemes quotidians en les seves aplicacions.

Tampoc hem previst mals usos que se'n puguin derivar essent una millora d'una tècnica molt concreta que, ja en si, no presenta riscos per a la societat. Ens basem en que no realitza tractament de dades confidencials, no hauria causar cap tipus d'addicció als usuaris, ni tenir cap tipus d'efecte perjudicial o nociu als usuaris finals o altres projectes que l'utilitzin.

Finalment, tampoc preveiem que el producte que volem desenvolupar en aquest projecte faci desaparèixer llocs de treball, ja que com hem dit és una millora d'una tècnia ja existent que per naturalesa és una tasca automàtica.

Els professionals més relacionats amb aquest projecte són els modeladors, que s'encarreguen de fer els models de malles amb gran detall. En cap cas la seva feina perilla a causa d'aquest projecte. Ja que no afecta en la seva feina, sinó al resultat de la mateixa. És a dir, continuaran sent necessaris modeladors que realitzin models, i així aquest puguin ser simplificats per l'algorisme que proposem, tal com ara mateix estan fent molts altres algorismes de simplificació.

8. Conclusions i ampliació del projecte

Amb els resultats anteriors podem concloure que hem aconseguit una tècnica que és capaç de tractar models de malles que les altres tècniques actuals tenen greus problemes per a obtenir simplificacions acceptables.

Bona part de l'èxit o el fracàs de la simplificació recau en el pas de la reconstrucció de la superfície, obtenint els millors resultats en models amb molta geometria i molt densa, ja que l'*alpha shape* pot generar una millor superfície envolupant.

De cara a ampliar aquest projecte i continuar el treball realitzat, seria interessant provar diferents tècniques en cada pas de l'algorisme i comparar-ne els resultats. Per a poder comprovar quina és la més adient en cada pas o si hi ha tècniques millors per a diferents tipus de models. Per exemple, podem substituir el pas de l'*alpha shape* pel *ball-pivoting* per a intentar millorar l'eficiència d'aquest pas.

L'algorisme també té diversos punts febles que hem detectat i que es poden millorar, tal com ara detallarem.

El primer i principal problema sobre la tècnica és que actualment la realitzem a través de programes de tercers essent així un pipeline altament ineficient que limita de forma dràstica l'aplicació de la tècnica. Per tant, la millora obvia és implementar nosaltres mateixos tots els passos de l'algorisme utilitzant, per exemple, les llibreries *CGAL*[6] i *OpenMesh*[7].

Començant pel preprocessat que realitzem amb el programa *Graphite,* amb la normalització dels triangles del model a tractar. Aquest pas es pot substituir per una subdivisió de triangles més audaç. Per exemple, un triangle es pot subdividir i re-triangular recursivament fins que el valor d'*alpha* amb el que s'aplica la tècnica no pugui generar un forat a la malla a causa de la mida de la cara.

Així evitem normalitzar tota la malla, estalviant així temps de computació i la deformació al model que provoca la normalització de cares de *Graphite*. D'aquí podem comprovar que aquest pas en l'algorisme que em proposat és, tal com hem indicat opcional, ja que hi haurà valors d'*alpha* que no necessiten aquest preprocessat.

En el pas del càlcul de l'*alpha shape* del model, El programa realitzat en R ha resultat estar limitat a causa de la memòria interna de R que ens ha impedit treballar amb models de grans dimensions. Una implementació en C++ de ben segur ens hauria donat millors resultats en eficiència.

Deixant de banda la implementació de l'algorisme, aquest tampoc és perfecte. Hem vist diversos models en els quals no hem aconseguit bons resultats. Això es deu en part al fet de que el valor d'*alpha* que s'aplica al model és constant per a tot ell, quan en realitat hi pot

haver fragments del model que requereixin un valor d'*alpha* més gran i d'altres un valor d'alpha més petit. Com hem indicat al cas de la simplificació del model *cupboard*.

També hem detectat que l'algorisme que proposem funciona pitjor en models poc densos. En aquests models l'*alpha shape* que s'obté és o bé massa proper al *convex hull* del model o queden massa "sortints" a la superfície de l'*alpha shape*.

En el primer cas l'*alpha shape* del model han perdut tanta geometria que la semblança amb el model original és mínima. Apart, un cop s'apliquen les tècniques de *baking* per a mapejar les normals i les textures del model original la quantitat de geometria sobrant provoca una deformació apreciable al mapeig. Per exemple el cas del model "House Plant" i algunes zones del "Stalks of corn".

En el segon cas tots els "sortints" que hi ha a la superfície del l'*alpha shape* generen molta superfície, creant greus deformacions. És el cas del model "Stalks of corn", que té diverses fulles que sobresurten de forma molt abrupte.

Bibliografía

[1] Luebke, D., Reddy, M., D. Cohen, J., Varshney, A., Watson, B., & Huebner, R. (2003). *Level of Detail for 3D Graphics, Chapter 2 Mesh Simplification*. Berkeley: Morgan Kaufman Publishers.

[2] Garland, Michael, and Paul S. Heckbert. (1997). *Surface simplification using quadric error metrics*. Proceedings of the 24th annual conference on Computer graphics and interactive techniques. ACM Press/Addison-Wesley Publishing Co..

[3] Shene, C. K. (2010). *Mesh Simplification*. Michigan Technological University. Retrieved from http://pages.mtu.edu/~shene/COURSES/cs3621/SLIDES/Simplification.pdf

[4] Fischer, K. (2000). *Introduction to Alpha Shapes.* Stanford Computer Graphics Laboratory. Retrieved from https://graphics.stanford.edu/courses/cs268-11-spring/handouts/AlphaShapes/as_fisher.pdf

[5] Lafarge, Thomas and Pateiro-Lopez, Beatriz. (2017). alphashape3D. Retrieved from https://cran.r-project.org/web/packages/alphashape3d/index.html

[6] CGAL. (n.d.). Retrieved from https://www.cgal.org/

[7] OpenMesh: Mesh Decimation Framework. (n.d.). Retrieved from https://www.openmesh.org/media/Documentations/OpenMesh-7.0-Documentation/a03920.html

[8] Hausdorff Distance. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Hausdorff_distance

[9] Part 4: Other methods for simplification. (n.d.). Retrieved January 16, 2019, from http://polygon-reducer.pc-guru.cz/other-methods-for-simplification?skin=tisk

[10] Cignoni, Paolo, Claudio Rocchini, and Roberto Scopigno. (1998). *Metro: measuring error on simplified surfaces*. Computer Graphics Forum. Vol. 17. No. 2. Oxford, UK and Boston, USA: Blackwell Publishers.

[11] Borodin, Pavel, Gabriel Zachmann, and Reinhard Klein. (2004). *Consistent normal orientation for polygonal meshes.* Computer Graphics International*, 2004. Proceedings*. IEEE.

[12] Hrádek, Jan. (2003). *Methods of surface reconstruction from scattered data*. Technical Report. Department of Computer Science and Engineering University of West Bohemia in Pilsen.

[13] Szilvśi-Nagy, M., and G. Y. Matyasi. (2003). *Analysis of STL files*. Mathematical and computer modelling 38.7-9: 945-960.

[14] Duncan Murdoch, M. (2018). *Package "rgl" 3D Visualization Using OpenGL*. Retrieved from https://r-forge.r-project.org/projects/rgl/

[15] About — blender.org. (n.d.). Retrieved January 18, 2019, from https://www.blender.org/about/

[16] ALICE project-team. (n.d.). Retrieved January 18, 2019, from http://alice.loria.fr/index.php/home.html

[17] Render Baking — Blender Manual. (n.d.). Retrieved January 18, 2019, from https://docs.blender.org/manual/en/latest/render/blender_render/bake.html

[18] Vaidyanathan, S. (2004). *Level of Detail Techniques*. Carnegie Mellon Graphics. Retrieved from http://graphics.cs.cmu.edu/nsp/course/15-462/Spring04/slides/LOD.pdf

[19] Hoppe, Hugues. (1996). *Progressive meshes.* Proceedings of the 23rd annual conference on Computer graphics and interactive techniques. ACM.

[20] Lévy, Bruno, and Yang Liu. (2010). *L p Centroidal Voronoi Tessellation and its applications*. ACM Transactions on Graphics (TOG). Vol. 29. No. 4. ACM.